INTEGRATING SYNTAX, SEMANTICS, AND DISCOURSE
DARPA NATURAL LANGUAGE UNDERSTANDING PROGRAM

R&D FINAL REPORT
Unisys Defense Systems
Contract Number: N00014-85-C-0012

Volume I -- TECHNICAL REPORT

ARPA ORDER NUMBER: 5262
PROGRAM CODE NO. NR 049-602 dated 10 August 1984 (433)
CONTRACTOR: Unisys/Defense Systems
CONTRACT AMOUNT: 1,704,901
CONTRACT NO: N00014-85-C-0012
EFFECTIVE DATE OF CONTRACT: 4/29/85
EXPIRATION DATE OF CONTRACT: 9/30/89
PRINCIPAL INVESTIGATOR: Dr. Lynette Hirschman PHONE NO. (215) 648-7554

SHORT TITLE OF WORK: DARPA Natural Language Understanding Program

REPORTING PERIOD: 4/29/85-9/30/89

TABLE OF CONTENTS

1 SUMMARY: MESSAGE UNDERSTANDING AT UNISYS 1

2 BACKGROUND 2

3 OBJECTIVES 3

4 ACCOMPLISHMENTS 3

5 OVERVIEW OF PUNDIT 5

6 APPLICATIONS 8
6.1 Message Processing 9
6.2 Natural Language Query Processing 9
6.3 Text Processing 13

7 SYNTACTIC COVERAGE 13
7.1 The Restriction Grammar Framework 14
7.2 Coverage of the Grammar 16
7.3 The Intermediate Syntactic Representation 19

8 THE LEXICON 22
8.1 The PUNDIT Lexicon 22
8.2 Lexical Coverage 23
8.3 Shapes 24
8.4 Lexical Tools 25

9 SELECTION 26
9.1 Overview of SPQR 26
9.2 Methodology 27

10 SEMANTICS 28
10.1 Predicating Expressions 30
10.2 Adapting the Analysis Process to New Predicating Expressions 30
10.3 Conjunction 31
10.4 Noun Phrases and Prepositional Phrases 33
10.5 The Semantics Rule Editor 37

11 TIME 38
11.1 Sentence Type 38
11.2 Intra-sentential Temporal Relations 39
11.3 Tense 39

11.4 Perfect and Progressive 39
11.5 Lexical Aspect 40
11.6 Adverbial Expressions 41

12 PRAGMATICS 41
12.1 Reference Resolution 42
12.2 The IDR 44

13 KNOWLEDGE REPRESENTATION AND REASONING 45
13.1 The Knowledge Base 47
13.2 PUNDIT Interface to M-Pack 47
13.3 Pfc -- The Reasoning Component 49

14 CROSS-COMPONENT PHENOMENA 50
14.1 Fragments 51
14.2 Nominalisations 52
14.3 Semantic Raising 52

15 EVALUATION OF NATURAL LANGUAGE PROCESSING 53
15.1 Natural Language Evaluation Workshop 53
15.2 MUCK-II 54
15.3 Evaluating Parsing in the Resource Management Domain 57

16 PARALLELISM 58

17 IMPLEMENTATION 60

18 FUTURE PLANS 60

19 ACKNOWLEDGEMENTS 61

20 BIBLIOGRPAHY FOR PUNDIT SYSTEM 63

TABLE OF FIGURES

Figure 1. Organisation of PUNDIT 6
Figure 2. Sample CASREP and Automatically Generated Summary 10
Figure 3. Processing a RAINFORM Message 11
Figure 4. Sample Trident Message and DB Fill 12
Figure 5. Templates Generated for MUCK-II 13
Figure 6. Sample Medical Abstract 14
Figure 7. Object Options in PUNDIT 17
Figure 8. Meta-Rule Applied to the Inr (Noun Phrase) Construct 18
Figure 9. ISR for *I found they have been reliable* 20
Figure 10. ISR for *I found them to have been reliable* 21
Figure 11. Partially Evaluated ISR 22
Figure 12. Lexicon for PUNDIT Core and Various Domains 24
Figure 13. ISR for the sentence *the engineer repaired the sac* 27
Figure 14. The Range of Syntactic Environments for Predicating Expressions 32
Figure 15. Sample Entry in the SRE 37
Figure 16. IDR 46
Figure 17. Sample Pfc Rule 50
Figure 18. Parsing Results for the Resource Management Domain 58

1. SUMMARY: MESSAGE UNDERSTANDING AT UNISYS

Message understanding systems are key to handling the steadily increasing volume of information required for intelligence, data fusion, logistics and maintenance operations. The capture and "understanding" of information in narrative text and messages is the focus of this contract. The goal is to convert this information into a representation of its meaning, which can be used to update a database with new information derived from the incoming messages, to summarize key information, or to issue appropriate commands to a back-end, e.g. to query a database or an expert system. The deliverable under this contract is the PUNDIT[1] natural language understanding system.

The PUNDIT natural language system is the most ambitious natural language understanding system built to date. Its goal is the integration of multiple linguistically-based knowledge sources (syntax, semantics, pragmatics), together with domain-specific knowledge (a domain model), to understand messages in a restricted domain. PUNDIT has been designed with portability as a major goal. To facilitate porting the system to new domains, PUNDIT maintains a careful separation of domain-independent information from domain-specific information. The system consists of a core domain-independent portion (20K lines of Prolog code) which remains constant from application to application. The domain-specific data constitutes an additional 5K-15K lines of code, depending on the application (20-40% of the total system in any given application); this includes the lexicon, semantics rules, co-occurrence constraints, domain model and discourse model.

Over the four years of this contract, we have built and demonstrated an integrated, linguistically-based message understanding system which provides syntactic, semantic and pragmatic processing, including analysis of temporal relations and computation of discourse referents. This system has been applied to the processing of multi-paragraph military message traffic, combining fixed field information with information from comment fields in "tactical message" (telegraphic) style. The system has been ported to four military domains (casualty reports, ship sighting reports, trouble failure reports for Trident submarine computer equipment, and Navy OPREPs), to database query applications, to medical abstracts, and to an air traffic control application.

Some of the highlights of our contributions over the past two years are:

- Leadership role in evaluation of natural language systems, organizing the first Natural Language Evaluation Workshop, and participating in MUCK-II.
- Port and demonstration on four military message domains and two query front-end applications.

[1]Prolog Understanding of Integrated Text

- Expansion of PUNDIT's syntactic, semantic and pragmatic coverage, to handle both query applications and multi-paragraph message understanding applications.
- Addition of modular interfaces for knowledge representation and reasoning components to PUNDIT, to draw inferences necessary for understanding discourse and generating a database updates or templates.
- Demonstration of a 10-fold speed-up of parsing on a 12-processor shared-memory system, through or-parallelism inherent in exploration of the parse search space.
- Release of PUNDIT to research groups in academia, government and industry under a no-cost license agreement; six licenses have been issued and more are being processed, for requests from universities, two industrial R&D organizations, and three government sites.

Under this contract, we have been able to demonstrate the feasibility of building a portable, linguistically-based message understanding system. Although many groups have examined the component modules in isolation, PUNDIT represents the first systematic attempt to integrate these modules into a coherent architecture for message understanding. In the course of this contract, we have demonstrated the applicability of this architecture to other language understanding applications, including query front-ends and the processing of non-message texts; we have also demonstrated its portability across a variety of domains. As the results of the MUCK-II Message Understanding Conference illustrate, message understanding technology is rapidly maturing. This work lays the groundwork for advanced applications in handling military messages, browsing through intelligence reports, facilitating retrieval of material stored in textual form, and for building natural language interfaces to complex systems.

2. BACKGROUND

This report summarizes the work done by Unisys under the contract *Integrating Syntax, Semantics, and Discourse.* This work was undertaken as part of the DARPA Strategic Computing Program in Natural Language Understanding. This report focuses primarily on the contract period May 1987 through September 1989, since there is a comprehensive report (R&D Status Report) covering the first two contract years, May 1985 through April 1987.

During the first two years of the contract, Unisys (then SDC) collaborated with NYU and exchanged a number of modules with the NYU researchers. This collaboration contributed to the progress of the both the NYU Proteus system, and the Unisys PUNDIT system. However, by the end of this initial two-year period, the systems had also diverged considerably in focus and in organization. As a result, the research during the second two years of the contract has been done largely independently and this report will focus exclusively on the Unisys progress and results. The initial sections of this report review the program objectives and overall accomplishments. Sections 5 and 6 provide an overview

of the system and describe the applications of PUNDIT to date. Sections 7 through 14 describe the coverage of the PUNDIT system, with special emphasis on features added to PUNDIT over the past two years. We conclude with a section on *Performance Evaluation*, and a brief summary of on-going work on parallelism in natural language processing. In the section *FUTURE PLANS*, we outline some of the open research issues for continued work in message processing, and the relation of the message understanding work to our current work in Spoken Language Systems. The final two sections provide information on PUNDIT's implementation, and acknowledgement to the many people who have contributed to this work.

3. OBJECTIVES

The objectives of the contract were to demonstrate *domain-specific text-understanding*, as described in the Strategic Computing Plan: "*understand[ing] streams of text to achieve automatic input of information transmitted in that form*", (p. 37, [1]). Specifically, we proposed to focus on the construction of a system consisting of domain-portable, integrated syntactic, semantic, and discourse components to perform this processing.

During the first two years, this system was to be be developed in the context of work on a single limited domain or *sublanguage*, and during the second two years of proposed work, the portability and increased robustness of the system was to be demonstrated by applying it to a new military domain.

The technical objectives were to demonstrate:

- Coverage and Reliability
- Portability and Maintainability
- Performance and Improved Analysis Algorithms.

The performance goals were:

- Correct syntactic and semantic analysis of unseen input at a rate of 65% (75% for documents where all the words are known in advance, with a false success rate of less than 3%).

4. ACCOMPLISHMENTS

This section summarizes our accomplishments against the objectives stated in the Statement of Work.

Integration of Syntax, Semantics, Pragmatics

- Development of modular, interleaved system architecture, enhancing portability by enforcing segregation of core PUNDIT from domain-specific modules.
- Integration of PUNDIT with knowledge representation and reasoning

systems to provide inferencing capabilities.

Technology Transfer

- System installed at National Library of Medicine, Unisys applications group;

- System available under no cost license agreement; requested by RADC, CECOM, Harvard, U., Lehigh, Penn., Cambridge U., Swedish Inst. of Computer Science, SAIC, and several other organizations.

Coverage and Reliability

- Treatment of fragmentary input as found in message traffic, by means of minimal extensions to the syntactic, semantic and pragmatic components.

- Semantic coverage of nominalized verbs, adjectival participles and noun predicates;

- Processing of intra-sentential temporal information, including lexical aspect, tense, adverbial phrases;

- Processing of referring expressions (including definite and indefinite noun phrases, phrases with omitted determiners, and reference to events).

- Handling of complex multi-paragraph message formats, including integration of fixed field and narrative field data.

- Meta-rule treatment of wh-constructions, providing a general, efficient treatment of wh-constructions which has been integrated with a general meta-rule treatment of conjunction.

Portability and Maintainability

- Port to four military message domains (casualty reports, ship sighting reports, trouble failure reports for Trident submarine computer equipment, and Navy OPREPs);

- Port to two database query applications, for querying databases about ships and their locations;

- Port to handling of complex journal-quality prose in medical abstracts;

- Port to two spoken language applications: an air traffic control

application, capturing controller/pilot exchanges; and a front-end for a program for providing directions in Cambridge, MA.

- Creation of a *System Administration Procedure* (SAP), to support system testing and incremental updating of the system simultaneously in multiple domains; this supports backporting general system improvements to previously developed domains (see documentation in Volume II of this report).

Performance and Improved Analysis Algorithms

- Interactive selection component, interleaved with syntax, uses semantic (selectional) information to filter parses, producing dramatic, 6-fold, decrease in number of parses.

- Implementation of a Dynamic Translator, combining interpreted rule-pruning with rule-compilation, to produce a 20x speed-up over interpreted code without rule pruning.

- Speed-up of 10x using 12 processors, by exploiting or-parallelism inherent in parsing.

Performance Goals

- Demonstrated PUNDIT on Resource Management domain, in preparation for Spoken Language Understanding; 85% of the training corpus and 76% of the test corpus received a correct parse.

- Successful participation in the MUCK-II Message Understanding Conference, porting PUNDIT to Navy OPREPs domain, for template filling application (results not publicized by agreement of the workshop participants).

5. OVERVIEW OF PUNDIT

The organization of PUNDIT is illustrated in Figure 1. The components of this system are best illustrated by considering how they apply to an input text. The initial input to PUNDIT is a string of characters. The *lexical processor* assembles the characters into tokens and associates with each token a definition that is either found in the *lexicon* or can be inferred from the structure of the tokens (*shapes*). The lexicon and shapes represent the *data* to the lexical processor and contain some domain-specific information; the lexical processor mechanism is entirely domain-independent.

Syntactic processing in PUNDIT yields two syntactic descriptions of the sentence. The *parser* constructs an extremely detailed analysis of surface structure. This surface tree is regularized into an operator-argument notation called the *Intermediate Syntactic Representation*, or ISR. The ISR is a regularization of the surface syntactic parse into a canonical predicate-argument form

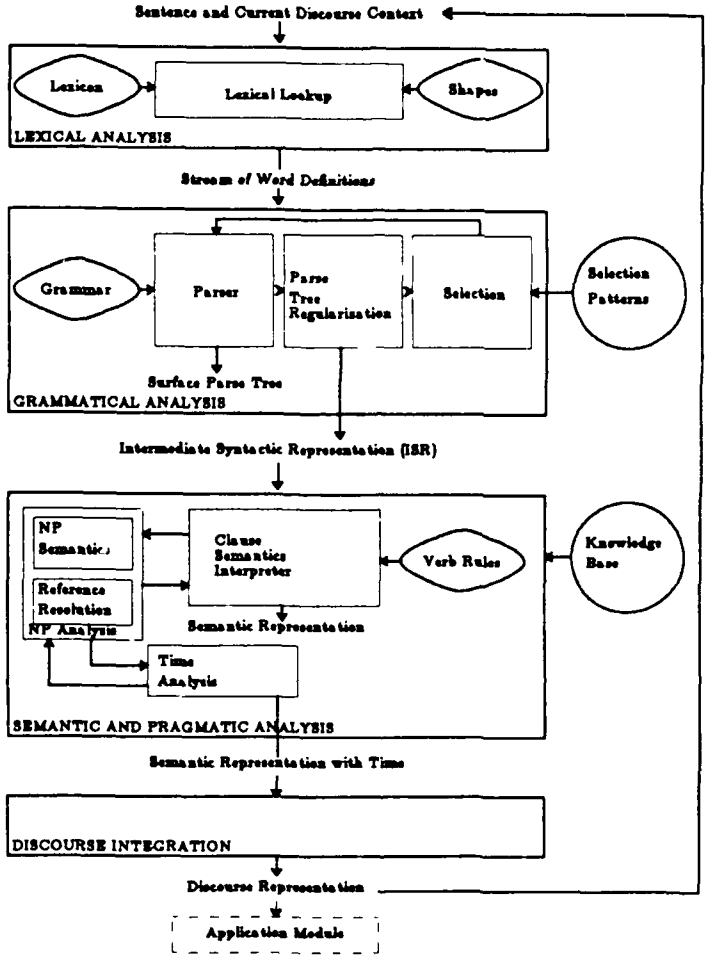

Figure 1.
Organization of PUNDIT

appropriate as input to the semantics and selection components. In the ISR, structural information not relevant to other modules is stripped away or expressed in a vocabulary appropriate to other modules. For example, a complex verb-auxiliary structure such as *harpoons had been launched* is regularized into the list (simplified for presentation):

```
[past, perfect, launch, subj(passive), obj(harpoons)]
```

In addition, the ISR represents an expansion of the surface parse in that certain kinds of information are made more explicit in the ISR: The scope of modifiers

under conjunction, for example, is made explicit in the following ISR (which is simplified here for clarity):

```
[past, and,
    [enter, subj(aircraft),obj(area)],
    [take,subj(passive),obj(aircraft),pp(by escort)]]
```

Fragmentary input is expanded to the extent that this expansion is syntactically determined. A fragment such as *badgers inbound* is expanded to incorporate a copula:

```
[untensed, be, subj(badgers), adj(inbound)]
```

The parser and the ISR translator are both domain-independent procedures; the *grammar*, consisting of context-free BNF definitions and restrictions on the shape of the parse tree, represents the data to these components. Although there are domain-specific aspects of the grammar rules, they require only very limited changes from domain to domain.

The ISR is the input to the *selection* component SPQR, whose function is to block semantically anomalous parses before they are sent to the semantic component. The selection mechanism interacts with the user to create a database of lexical co-occurrence patterns; the ISR of a sentence is checked against this database upon completion of each clause and noun phrase. The mechanism itself is domain-independent, but the set of selectional patterns it creates contains much domain-specific information. The ISR is also the input to the semantics analysis.

The *semantic interpreter* draws upon a database of lexical semantic rules to decompose verbs (and other predicates, such as nominalizations and adjectives) into more basic predicates and fill their semantic roles with syntactic constituents. These rules express the decompositions of specific verbs, the semantic constraints on potential fillers of their thematic roles, and any verb-specific rules for the syntactic expression of arguments.

The semantic component interacts with two pragmatic components to construct this representation. The first, *reference resolution*, provides specific referents for the noun phrases that are proposed fillers for semantic roles and provides fillers for semantic roles which are not realized syntactically. The *temporal* component interacts recursively with the semantic interpreter to provide a temporal analysis of the sentence: It assigns an actual time to the predication, if appropriate; it classifies the situation denoted by the predication as an event, state, or process; and finally, it computes the relative temporal order of distinct situations mentioned within the same sentence where the information is provided explicitly. The events, states, and processes created by the time component become discourse entities, which can be subsequently referred to.

Domain knowledge is represented in M-Pack, a semantic-net based knowledge representation system which was developed at the Paoli Research Center and is similar to KL-ONE[3]. The selection, semantics, and pragmatics components

consult this component as needed to support their processing.

The output of semantics and pragmatics is the *Integrated Discourse Representation*, or IDR, which has been used to underlie a variety of previous applications such as a tabular summary or automatic generation of database entries or queries. The IDR is a very detailed representation of the content of a text, but it does not represent the kind of inferences about the text that are needed for the OPREPs template-filling application.

In order to provide this ability, we have added a new component called Pfc (Prolog Forward Chaining) described in section 13, which provides an inference engine for forward chaining rules. Pfc was developed at the Paoli Research Center independently of PUNDIT. The IDR for each sentence is currently the input to Pfc. However, we also intend to explore the possibility of accessing Pfc during processing, at specific points where non-linguistic, domain-specific reasoning would be useful.

Over the last five years, over 20 person-years have gone into PUNDIT's development. PUNDIT development has been funded by DARPA, the National Science Foundation, the National Library of Medicine, and Unisys internal funding. PUNDIT has been used to process maintenance reports of air compressor failures (CASREPS), RAINFORM and OPREP messages of naval activities, and maintenance reports of equipment failure aboard Trident submarines, as well as database queries about ship status and locations. Currently in progress are treatments of medical journal abstracts about congestive heart failure, air traffic control dialogs, and an interface to an expert system for finding one's way around Boston. For these last two applications, we plan to integrate PUNDIT with a spoken-language capability.

6. APPLICATIONS

PUNDIT has been applied to a range of domains over the past four years. For a specific application within a given domain, the core system is augmented by domain-specific data files (the knowledge base, the lexicon, etc) and is front- and back-ended by application modules. These provide the user interface to PUNDIT, and may perform application-specific tasks based on PUNDIT's output: for example, summarizing the input, interfacing to an expert system, and updating or querying a database. All of these back-ends use the same IDR structure (the output of PUNDIT) as input.

The two major application areas to date have been message processing and database queries. Our goal has been to establish the portability of PUNDIT across different subject matter areas (intelligence messages, maintenance reports), and to establish its utility for a variety of tasks. We see the basic capabilities described here as applying to a wide variety of problems, including intelligent message routing, intelligent information retrieval, sorting and browsing through large text collections, intelligent front ends for databases and expert systems.

6.1. Message Processing

Applications have been developed to process four types of messages or reports: maintenance reports on Burroughs equipment, messages reporting equipment failures on Navy ships (CASREPs), Navy RAINFORM tactical messages, and 'trouble and failure reports' (TFRs) from Trident submarines.

The CASREPs application was developed in the context of a Navy battle management domain focused on force readiness. For this application, the system processes the remarks field of messages and generates a tabular summary of the major problems and findings. Figure 2 below shows a sample CASREP and the summary output generated by PUNDIT.

The next two applications were also in the message processing domain; these were the RAINFORM messages, for MUCK-I and a set of Trouble-Failure Reports, for computer equipment problems on board the Trident submarine. For MUCK-I, the goal was to process the RAINFORM messages. The RAINFORM messages are ship sighting messages, similar in content to the OPREP messages used in MUCK-II. A sample message in shown below in Figure 3, with an explanation of how PUNDIT reconstructs the missing information from context.

The Trident TFR application focused on handling a complex message format, consisting of fixed field data (used to set a "context" for the narrative), and a number of narrative fields, each with a header, eliciting a specific type of information (e.g., cause of failure)[2]. Figure 4 below illustrates this style of message. To simulate message capture at the point of message entry, we implemented an interactive front-end, which allows a message to be collected interactively through a prompt-response dialogue. The system asks the user a number of questions, designed to elicit key facts about the problem (what went wrong, what was the cause, what action was taken, etc.); PUNDIT analyzes, validates and integrates the user's answers into a representation of the message content. The results of this analysis are used to update a historical database of equipment problems, which can then be queried using a pre-defined query language. The most recent application of PUNDIT has been to the OPREP domain, for MUCK-II. We show below in Figure 5 the template fill created from an OPREP message.

6.2. Natural Language Query Processing

In query processing, the core components of PUNDIT analyze an English query and produce a set of meaning representations, which are then mapped to a set of database relations. For queries to a foreign database (either remote or local), the database relations are translated into QUEL, and the resulting QUEL query is used to access an INGRES database. The results are displayed to the user. Alternatively, the database relations may be used to access an integrated Prolog relational database. These approaches have been used to support English queries on an INGRES database of ship movements and on Prolog databases of

FAILURE OF ONE OF TWO SACS. UNIT HAD LOW OUTPUT AIR PRESSURE. RESULTED IN A SLOW GAS TURBINE START. TROUBLESHOOTING REVEALED NORMAL SAC LUBE OIL PRESSURE AND TEMPERATURE. EROSION OF IMPELLOR BLADE TIP EVIDENT. CAUSE OF EROSION OF IMPELLOR BLADE UNDETERMINED. NEW SAC RECEIVED.

Status of Sac:
Part: sac State: inoperative

Damage:
Part: blade tip State: eroded

Finding:
Part: air pressure State: lowered

Finding:
Part: oil pressure State: normal

Finding:
Part: oil temperature State: normal

Finding:
Agent: ship's force State: has new sac

Figure 2.
Sample CASREP and Automatically Generated Summary

Message number: 11
Enemy platform: SUBMARINE
Reporting platform: TEXAS
Report time: 0830T

Sighting message:
SIGHTED PERISCOPE AN ASROC WAS FIRED PROCEEDED TO STATION VISUAL CONTACT LOST, CONSTELLATION HELO HOVERING IN VICINITY.

PROCESSING HIGHLIGHTS

- Parsing run-on sentence.

- Message header creates context for interpreting the message:

TEXAS *sighted a periscope* [**of submarine1**].
An asroc was fired by **TEXAS** *at* [**submarine1**].
TEXAS *proceeded to station.*
Visual contact on [**submarine1**] *lost by* **TEXAS**.
Constellation helo hovering in vicinity.

Figure 3.
Processing a RAINFORM Message

TFR number: 1 Equipment code: TLS
Part: 123456 Name: tape lock switch
TFR date: 4/3/85 Report date: 4/3/85

A. First indication of trouble:
WHILE PERFORMING TEST 1, TAPE LOCK SWITCH WENT OFF LINE TO NOT READY.

Relation	Attribute	Value
header_info	tfr_number	1
	equipment_code	TLS
	part_number	123456
	tfr_date	4/3/85
	report_date	4/3/85
	speaker	J JONES
failure	tfr_number	1
	failed_component	[switch1]
	type_of_failure	
	cause_of_failure	
circumstances	tfr_number	1
	circumstance_of_failure	[test 1]
symptoms	tfr_number	1
	symptoms_of_failure	[off line]
procedures_ref	tfr_number	1
	procedure_name	test
	procedure_number	TEST 1
part_in_equipment	tfr_number	1
	part_number	123456
	equipment_code	TLS

Figure 4.
Sample Trident Message and DB Fill

```
 0.  MESSAGE ID                              DEV-GP1-N09722-008
 1.  EVENT: HIGHEST LEVEL OF ACTION          ATTACK
 2.  FORCE INITIATING EVENT                  HOSTILE
 3.  CATEGORY(S) OF EVENT AGENT(S)           SURF
 4.  CATEGORY(S) OF EVENT OBJECT(S)          NO DATA
 5.  ID(S) OF O-TH LEVEL EVENT AGENT(S)      HOSTILE FORCES
 6.  ID(S) OF O-TH LEVEL EVENT OBJECT(S)     USS STERETT CG-31
 7.  INSTRUMENT(S) OF O-TH AGENT(S)          SSN-12
 8.  LOC OF OBJECT(S) AT TIME OF EVENT       NO DATA
 9.  TIME(S) OF EVENT                        "0819(L)"
10.  RESULT(S) OF EVENT                      RESPONSE BY OPPOSING FORCE

 0.  MESSAGE ID                              DEV-GP1-N09722-008
 1.  EVENT: HIGHEST LEVEL OF ACTION          ATTACK
 2.  FORCE INITIATING EVENT                  FRIENDLY
 3.  CATEGORY(S) OF EVENT AGENT(S)           SURF
 4.  CATEGORY(S) OF EVENT OBJECT(S)          NO DATA
 5.  ID(S) OF O-TH LEVEL EVENT AGENT(S)      USS STERETT CG-31
 6.  ID(S) OF O-TH LEVEL EVENT OBJECT(S)     UR KIROV CLASS CGN
 7.  INSTRUMENT(S) OF O-TH AGENT(S)          HARPOON
 8.  LOC OF OBJECT(S) AT TIME OF EVENT       NO DATA
 9.  TIME(S) OF EVENT                        "0823(L)"
10.  RESULT(S) OF EVENT                      NO DATA
```

Figure 5.
Templates Generated for MUCK-II

analyzed CASREP and RAINFORM messages.

Recently, we have used PUNDIT to process queries in the Resource Management domain. Because no database was available, our experiments[18] were limited to collecting statistics on parsing the queries.

More recently, we have applied PUNDIT to processing exchanges between air-traffic controllers and pilots; also to queries to an expert system for navigating through Cambridge, MA.

6.3. Text Processing

Although our major focus has been on telegraphic message processing, PUNDIT is also able to parse full journal quality text. For example, it can process the medical abstract shown below, in Figure 6.

7. SYNTACTIC COVERAGE

This section provides an overview of PUNDIT's approach to syntax. PUNDIT constructs two syntactic representations for every sentence. The first

Title: *Clinical evaluation of teicoplanin for therapy of severe infections caused by gram-positive bacteria.*

Teicoplanin was evaluated in 47 patients with severe infections, including 14 patients with bone infections, 11 patients with soft-tissue infections, 7 patients with endocarditis, 5 patients with pneumonia, 3 patients with septic thrombophlebitis, 3 patients with septicemia of unknown origin, and 4 patients with miscellaneous infections. Overall, bacteremia was documented in 24 patients.

A total of 22 patients (46.8%) were clinically cured, 8 patients (17.0%) improved, 2 patients (4.3%) had relapses after initial improvement, and 15 patients (31.9%) failed to respond. The results were better in nonbacteremic patients (19 of 23 patients [82.6%] were cured or improved) than in patients with bacteremia (12 of 24 patients [50.0%] were cured or improved). Bacteriological cure occurred in 25 patients (53.2%), and superinfections were documented in 6 patients (12.8%). No major adverse effects were observed. We conclude that teicoplanin is a potentially effective and well-tolerated antimicrobial agent for therapy of nonbacteremic infections caused by gram-positive bacteria.

Figure 6.
Sample Medical Abstract

representation is a surface structure parse tree, based on string grammar[12]. The second representation is constructed from the first to produce an *Intermediate Syntactic Representation (ISR)* which regularizes, and in some cases expands the surface structure parse tree into a representation of just those aspects of syntactic structure that are relevant to semantics and selection. In this section, we first describe Restriction Grammar as a framework, then outline the grammar coverage, and then describe the ISR.

7.1. The Restriction Grammar Framework

The syntactic component uses Restriction Grammar, which is a logic grammar framework. Like most logic grammars, Restriction Grammar supports a grammar consisting of context-free rules (BNF definitions) and constraints or *restrictions*, to capture context-dependent relations [14]. However, Restriction Grammar also has a number of interesting features that distinguish it from many logic grammars.

Restrictions

Restriction Grammar, like other logic grammars, is based on the notion of parsing as a proof of sentencehood. However, restrictions in Restriction Grammar can be viewed as well-formedness constraints on the structure of the parse tree (or proof tree). Restrictions gather contextual information by examining the parse, and not by using parameter passing. Also, restrictions are constrained in their power. In particular, they can only reject or accept proposed analyses; they do not **add** information or instantiate variables (in contrast to, for example, unification grammars). These features make it easy to write meta-rules in RG and also to experiment with alternative control strategies (parsing strategies).

Meta-Rules

RG uses meta-grammatical notions to capture linguistic generalizations and to maintain compactness of the grammar. A set of meta-rules generates rules for conjoined structures from definitions of simple (unconjoined) constituents [15]. Wh-constructions are also treated via a different type of meta-rule [17]. For wh-constructions, grammar annotations in the BNF definitions indicate the need for a gap or the filling of a gap. The annotated grammar rules are processed into executable form, which inserts paired parameters for passing information on the state of the gap.

Regularization Rules

Coupled with each BNF definition is a regularization rule for computing the Intermediate Syntactic Representation (ISR) compositionally from the surface syntax. The compositional computation of the ISR permits interleaving of syntax and semantics, to provide for earlier application of semantic constraints during parsing. For both conjunction and wh-constructions, the rules to generate the ISR integrate easily into the meta-grammatical framework.

Parsing Control

Restriction Grammar is implemented in Prolog. A translator turns RG rules (context-free BNF definitions with restrictions) into executable Horn Clauses in a fashion similar to DCGs. The parser for RG can be readily implemented as a top-down parser, and this has been the parsing strategy in use for most of our applications. One refinement has proved very useful, namely the *Dynamic Translator*, which mixes interpreted and translated code, to allow grammar rules to be modified during parsing[8]. Because of the limited power of restrictions, we have also been able to use RG in a bottom-up, left-corner parser. In this implementation, if a restriction requires more of the parse tree than has been built, it is delayed (by use of a *continuation*). We have also explored the notion of delayed restrictions and continuations for parsing with well-formed substrings[9].

Or-Parallelism

Parsing is a search problem. The form of RG has made it possible to exploit the or-parallelism inherent in the set of alternative grammar rules. Simulation experiments have indicated that speed-ups of 20-30 fold are possible, given a sufficient number of processors (40-60 processors)[16, 19]. More recently, we have run a parsing application on a 12-processor system that showed as much as 11 fold speed-up.

7.2. Coverage of the Grammar

This subsection will summarize the current state of PUNDIT's coverage.

Noun Phrases

Coverage of noun phrases is generally very good. It includes treatment of complex pre-nominal modifiers: multiple nouns, adjectives, qn expressions such as *a two-foot deep hole*, and nq expressions, such as *the number 2 pump*. Nominalizations, e.g., *failure*, are handled as ordinary noun phrases in the syntax, so they are covered and later converted by semantics to capture the underlying verb semantics. A wide range of post-nominal expressions are also covered, including multiple prepositional phrases, participial expressions (*the book read by the students, the person running the race*), adjective expressions (*the student present for the exam*), appositives and parenthetical expressions (*Florence Joyner, the Olympic athlete*, and *my PC (the one I bought a month ago)*. Relative clause coverage has been greatly expanded with the introduction of the new wh-module and includes both standard relative clauses, and zero-complementizer relatives (*the person I saw*). Pronouns are handled by a separate **lpror** option for the noun phrase; this is done because pronoun take a highly restricted set of left and right adjuncts, compared to nouns.

Adjective and Adverbial Phrases

Coverage of adjective phrases, in pre-nominal position, predicative position and verb complement position is extensive. In predicative and verb complement positions, adjectives can take complex right modifiers, including prepositional phrases (*certain of a fact*) and a variety of clausal complements (*certain that they came, certain to come*). In the left adjunct slot, adjectives can be modified by adverbs, e.g, *very certain*. The coverage of adverbials in PUNDIT includes left and right modifiers and a recursive definition (e.g., for *very quickly*).

Verb and Verb Complements

Our current grammar includes more than forty classes of verb complement (object). Selection of the appropriate complement set is controlled by a pruning mechanism that takes the intersection of the verb's subcategorization constraints (given in the *objlist* for the verb entry in the lexicon) with the set of object options. Classes of complement types are listed in Figure 7 below. Each

direct object,
ditransitive,
objects of auxiliary verbs:
vo (*I may* **read the book**);
vingo (*I am* **reading the book**);
veno (*I have* **finished the book**);
venpass (*She was given the book*);
objects of *be* and other copulative verbs:
objbe (*They are here/at home; they remain leaders*),
direct object + prepositional phrase,
particle + various object types,
(e.g., *close up, close up the store, close the store up*),
clausal objects
(e.g., *I said that I would come; it seemed to be raining*).
equi-verb objects
(e.g., *I wanted to go*).
small clauses
(e.g., *they painted the house red*).

Figure 7.
Object Options in PUNDIT

of these object options has a regularization rule associated with it that allows correct reconstruction of the underlying semantics, including correct handling of subject/object control issues. This is done by the Intermediate Syntactic Regularization component and will not be further discussed here; see the PUNDIT Guide to Verb Objects (in Volume II) for more complete documentation of PUNDIT's object options. One respect in which PUNDIT's treatment of object differs from string grammar is in a uniform treatment of modals, which simply take the object option vo, namely infinitive verb + object.

Sentential Adjuncts

The grammar covers a variety of sentential adjuncts, including adverbial modifiers (adverbs and prepositional phrases), purpose clauses (*I did it to win*), and a range of subordinate clauses (*until finished; before they came; after running the race*). It now also covers a class of adverbial phrases consisting of a lone noun phrase. In normal English, this includes time expressions, e.g., *I left last week.* Also needed for message texts is a similar location adverbial construction, such as *lesion right lung*, where *right lung* is a locative phrase without a

preposition. Both of these require strong selectional or semantic constraints, in order to avoid taking almost any noun phrase in any adjunct slot. Not included yet are right-dislocated relative clauses (*the person came whom I wanted to meet*).

Conjunction

The conjunction meta-rule component generates rules to handle conjunction from the basic BNF definitions. Conjoining is allowed only at certain nodes, which eliminates some of the spurious ambiguity that can be associated with treatments of conjunction. The current mechanism handles a variety of conjunctions (*and, or, but*), paired constructions (*both...and, neither...nor*) and "comma-conjunction" (use of comma to take the place of an explicit conjunction in a list such as *apples, oranges and pears*). Since the meta-rule generates a recursive definition, an arbitrarily long series of conjunctions can be handled. Figure 8 illustrates the application of the meta-rule for the *lnr* construction (left-noun adjunct + head noun + right noun adjunct). Thus BNF definitions can be written without worrying about conjunction; the meta-rule component is then applied to generate automatically the correct rules to support optional conjoining.

In addition, the meta-rule component allows for gapping under conjunction. In particular, it can handle gapped subject, gapped object, and gapped verbs, as follows:

I mixed up the batter and baked the cookies.
I cooked and they ate the cookies.
I baked the cookies and Robin the cake.

Wh-Constructions

The new meta-rule component for wh-constructions now covers questions, relative clauses and indirect questions (*I don't know what they want*). We plan to

---

```
lnr ::= ln, nvar, rn

      => (via meta-rule)

lnr ::= ln, nvar, rn;
      ln, nvar, rn, conj_wd, lnr.
```

Figure 8.
Meta-Rule Applied to the lnr (Noun Phrase) Construct

---

extend it shortly to cover headless relative constructions (*Whatever you need is here*) as well. It supports the interaction between conjunction (and its gaps) and the wh-constructions (and their gaps).

In the wh-constructions, the function of the meta-rule is be to introduce parameters into each definition, so that gap information can be passed around, namely the need for a gap, or the fact that a gap has been found. This makes the handling of wh-constructions invisible to the grammar writer, who need only worry about routine constructions. A meta-rule treatment has several advantages over a treatment via interpreter in the style of Extraposition Grammar. These include appropriate limitation of the scope of the gap, the ability to translate/compile the grammar rules, and ease of integration with conjunction.

Wh-constructions are handled by parameterizing the grammar rules: top-level rules that license a gap are assigned parameters by hand, as are the terminal gap-accepting rules (e.g., realization of a noun phrase as a gap). Meta-rules then generate appropriate "pathways" (via parameters) that license gaps for just those elements that can dominate a gap. Parameters are paired; the input parameter licenses the gap, and the output parameter indicates that the gap has been found (and filled). This "change of state" in the paired parameters is used to ensure that each gap is filled once and only once. The conjunction meta-rule then operates on the parameterized wh-rules to link gaps within conjoined structures by unification, so that any gap within a conjoined structure is treated identically for all conjuncts.

Fragments

Because much of our work has been focused on message traffic, PUNDIT supports a comprehensive, elegant treatment of fragmentary and run-on sentences that are characteristic of message text[21]. There are five basic fragment types, including fragments for missing subject (**tvo:** *was repaired*), missing verb (**zero_copula:** *disk bad; disk repaired*), missing subject and verb (**predicate:** *broken since yesterday*), missing object (*engineer repaired*), and noun phrase fragment (**nstg_frag:** *bad drive*). Other recently added center string rules include rules for response fragments, necessary to handle certain kinds of question/answer interchanges, e.g., *Are you going? Yes.*

7.3. The Intermediate Syntactic Representation

While the level of complexity of the surface structure parse tree is necessary for doing correct syntax, not all of that complexity is necessary for further semantic and pragmatic processing. In order to buffer PUNDIT's semantic and pragmatic components from the complexity of the surface structure parse tree, we developed the *Intermediate Syntactic Representation* (ISR), which provides a more canonical representation of the information expressed in the parse tree. The ISR is an operator/operand form that regularizes the predicate-argument-modifier relations implicit in the parse tree. In addition, the ISR provides a

clean separation between syntax and semantics, allowing changes in the grammar to be made independently of semantic changes.

An example of an ISR expression is given in Figure 9 for the sentence *I found they have been reliable.* The ISR is represented as a Prolog term. The top-level operator of this ISR is the tense marker 'past' which is an operator that takes one argument (an OP1). Using Prolog list syntax, if the first element of the list is an OP1, then its argument is the remainder of the list. All tense and aspectual markers are OP1's. The first element of the remainder of the list is the verb *find.* Since verbs have variable numbers of arguments plus optional modifiers, the remaining elements of the list are taken to be the arguments of the verb, followed by its modifiers, if any. Here, the subject and object are verb arguments. The subject is the singular pronoun `I`, and the object is itself a clause, with OP1's `present` and `perf` indicating tense present and aspect perfect, and verb `be`. This ISR can be compared with a similar one for the sentence *I found them to have been reliable*, shown in in Figure 10. Here the differences in syntactic structure which are not relevant to semantic and pragmatic processing have been removed. The only difference in the ISR of these two sentences is in the tense marker of the object clause.

In order to construct the ISR, every rule in the grammar is annotated with an expression in the *Translation Rule Language* (TRL). These TRL expressions declaratively indicate how the ISR of a node should be constructed from the ISR of its children. The ISR's of terminal nodes are derived from the lexical entries of the words attached. Thus, the construction of the ISR is strictly compositional. The declarative nature of the Translation Rule Language also allows it to integrate smoothly with the meta-rule component.

Evaluation of the ISR goes on in two stages. In the first stage, TRL expressions are evaluated, which gives rise to a list structure representation for each node.

```
[past,
   find,
      subj([pro([i,singular,A])]),
      obj([present,
             perf,
                be,
                   subj([pro([they,plural,B])]),
                   adj([reliable])])]
```

Figure 9.
ISR for *I found they have been reliable*

```
[past,
   find,
      subj([pro([i,singular,A])]),
      obj([untensed,
             perf,
                be,
                   subj([pro([they,plural,B])]),
                   adj([reliable])])]
```

Figure 10.
ISR for *I found them to have been reliable*

These structures, however, may contain lambda expressions whose values may not be available at the current node. Thus, a second stage *simplification* is needed in order to perform lambda application and to remove unnecessary list structure.

TRL expressions are associated with grammar rules using a Prolog infix operator -> which has the body of a BNF expression on its left, and a TRL expression on its right. For a simple example, consider the grammar rule for object when the object of the verb is a simple noun phrase.

```
object ::= np
     -> lambda(Verb,lambda(Subject,[Verb,subj(Subject),obj(np)])).
```

The TRL expression here is a function of two arguments (actually, two functions of one argument each). The body of the lambda expression constructs a predication whose operator is the verb, with the subject and object as operands. The subj label wrapped around the subject inserts a syntactic role marker in the ISR. These markers are called 'semlabels' (semantic labels) because they are used in semantic processing to map syntactic constituents to semantic roles. The np atom in the TRL expression is a command to insert the ISR of the np child into the ISR of the object node at this point. When the object rule is completed, its ISR will not be simplifiable because the arguments to the function will not be available. One parent node of object is assertion, whose (simplified) grammar rule is:

```
assertion ::= subject, ltvr, object -> [object, ltvr, subject].
```

For the sentence *I found them*, evaluating the TRL expressions for assertion, subject, verb, and object yields the structure shown in Figure 11. The use of the list notation in the ISR is overloaded, indicating both operator/operand structure and lambda application. In this case, because the first element of the list is a lambda expression, we know that this is an instance of lambda application. The ISR of the object, a function of verb and subject, is applied to the

```
[lambda(Verb,
    lambda(Subject,
       [Verb,
          subj(Subject),
          obj([pro([they,plural,B])])])),
 find,
 [pro([i,singular,A])]]
```

Figure 11.
Partially Evaluated ISR

ISR's of verb and subject, to produce an operator-operand representation of the clause:

```
[find,
    subj([pro([i,singular,A])]),
    obj([pro([they,plural,B])])]
```

8. THE LEXICON

In describing lexical coverage, we distinguish between syntactic and semantic coverage. The syntactic properties of lexical items can potentially be exhaustively enumerated while semantic and pragmatic properties (i.e., *meaning*) are not enumerable. For example, proper names, which can be considered part of world knowledge, are not enumerable. Also, distinguishing among the different uses of polysemous words depends heavily on world knowledge and can be highly domain dependent. This section describes syntactic lexical coverage.

8.1. The PUNDIT Lexicon

The PUNDIT lexicon has several features that are relevant to this discussion.

Entries indexed on first word

Each lexical entry is entered into the (Prolog recorded) database, indexed on the *first word*. Most entries, of course, have only one word; however, for multi-word expressions (e.g., *red blood cell*), the entry is indexed only on the first word (*red* in this example). Each entry in the lexicon consists of the WORD, the index term, the root, and the attribute list. The source form of the lexicon looks as follows:

```
:(WORD, root: ROOT, ATTRIBUTE_LIST).
```

where ATTRIBUTE_LIST is a list of the form:

```
[LEXICAL_CLASS : ATTRIBUTES | MORE_ATTRIBUTES].
```

Idioms (multi-word expressions) are entered by use of the circumflex infi operator (^), which connects the words in the multi-word expression, e.g.,

```
:(red^blood^cells, root: red^blood^cell, [n: [ncount1, plura
```

Compression of redundant information

The PUNDIT lexicon enters each morphological variant as a separate entry, since there is (currently) no separate morphological component. As a result, there is a great deal of redundancy between morphologically related entries. To minimize this redundancy, the lexicon compresses information, storing the full set of attributes in the *root* entry, and using pointers to this information in the morphological variants. This means that at lexical look-up time, the look-up procedure must "reconstitute" entries for individual words into their full form.

Multiple Entries

A single word may have multiple entries in the lexicon. This can reflect incremental additions to the lexicon, or it can reflect differing forms, e.g., different parts of speech, as in the noun *train* vs. the verb *train*; it can result from genuine homographs, such as the verb *can* used as a modal (*be able*) or as a transitive verb for the canning process. At times, it can also reflect an error, where two people have independently entered the same word into the lexicon. In any case, one function of the lexical look-up procedure is to amalgamate these entries into a single entry for purposes of parsing. Where two entries are identical, the program is smart enought to simply collapse them. In other cases, the union of the attributes is recorded.

Shapes: a grammar for productive forms

The last issue concerns the problem of how to store productive forms in the lexicon. This arises, for example, for numbers, dates, times, part numbers, etc. The solution in PUNDIT is to use a shapes grammar, which parses the tokens within a productive form, identifies the class (and attributes) of the lexical entry from the shape of its tokens, and assigns it a definition on this basis. Definitions derived from the shapes component are then added to the list of possible definitions for a word. This is discussed in more detail below.

8.2. Lexical Coverage

Our lexicon files contain entries for words giving their syntactic categories, the types of arguments they subcategorize, and the root form. Morphological variants for each word are listed separately with a field indicating the root. Coverage of the syntactic lexicon can be evaluated in terms of the following categories.

Major word classes
  Coverage of closed class items
    pronouns

determiners
prepositions
conjunctions
quantifiers
Coverage of open class items
verbs
nouns
adjectives
adverbs

The lexical entries for PUNDIT are organized in several lexicons. Associated with the core, domain-independent files there is a standard lexicon of entries with the potential to turn up in any domain. Associated with each domain is an add-on lexicon of specialized vocabulary. Figure 12 indicates the size of the various lexicons at the end of 1988. The total number of entries, which includes all morphological variants is given first, and then broken down into distinct roots, and 4 major syntactic categories. Note that a single lexical item can be cross classified. The column headed CORE represents the domain-independent lexicon. Three of the domains are labelled by the type of message: Trouble Failure Reports (TFRs), Casualty Reports (CASREPs), RAINFORMs (naval sighting messages); Ship DB is for an prototype application involving queries to a ships database; NLM is for National Library of Medicine. Since these figures were compiled, we have added three domains, and more than doubled the size of the core lexicon.

8.3. Shapes

The shapes component of PUNDIT, which functions as an extension of the lexicon, is designed to recognize tokens on the input stream which cannot all be entered in the lexicon, but whose definition can be inferred from their "shape".

	CORE	TFRs	CASREPs	RAINFORMs	Ship DB	NLM
Entries	1170	170	502	296	55	192
Roots	570	54	209	101	20	78
Verbs	153	13	34	27	1	13
Nouns	163	27	148	44	19	34
Adjs	152	14	33	8	-	29
Advs	101	6	24	-	-	14

Figure 12.
Lexicon for PUNDIT Core and Various Domains

Typical shapes from a real-world domain might include numbers, telephon numbers, social-security numbers. Social security numbers, for example, can be recognized as the following sequence of input tokens: three-digit-number, hyphen, two-digit-number, hyphen, four-digit-number. The shapes component is currently implemented as a definite-clause grammar operating on the input token stream, and is invoked by lexical lookup whenever a token is not found in the lexicon.

Some of the typical shapes currently handled by PUNDIT are the following:

integers, which obviously consist of simply an integer;
real numbers, which consist of an integer followed by period
followed by an integer;
fractions, which consist of an integer followed by a slash ("/")
followed by an integer;
dates, which consist of an integer (the month) followed by a slash
followed by an integer (the date) followed by a hyphen
followed by an integer (the time of day)
part numbers, which consist of a four-digit integer followed by a hyphen
followed by a four-digit integer

Certain other shapes, in particular those from the RAINFORMs domain, examine the internal structure of the input tokens. For example, in the RAINFORM domain, a time shape can consist of (among other possibilities) an atom of the form NNNNt or NNNz, where "N" can stand for any single digit.

8.4. Lexical Tools

This section describes the current Lexical Entry Procedure, and other lexical tools that have been developed. The Lexical Entry Procedure has been designed to provide consistency, completeness, and speed of entry for new words. The procedure elicits relevant linguistic information from the user, computes dependencies between attributes, and prompts for morphologically related forms (offering a "guess" as to the correct form). The program then automatically creates a set of related dictionary entries, with as much structure-sharing among the entries as possible. Before the entries are actually entered in the database or written to a file, the user may inspect and edit any entries created. Detailed documentation for the current Lexical Entry Procedure is available in the document, *PUNDIT Lexical Entry Procedure User's Guide*, in Volume II of this report.

A number of tools have also been developed for building and managing dictionaries. These include:

Concordance Program

The concordance program creates a concordance for a corpus of text. As an intermediate step, it creates a file with the words that are in the text but not in the lexicon.

Merging Dictionary Tool

A tool exits for merging dictionaries: *dictMerge* takes a list of lexicon files and merges them to make one dictionary. It also checks that there are no redundant entries.

Sorting

When it is known that there is one lexical entry per line in a lexicon file, the unix utility **sort** can be used to merge dictionaries. It should be used with the **-u** (unique) and **-d** (dictionary order) parameters.

Missing Words

unknown_words.pl: determines which words of a corpus are not in a lexicon that has been loaded into an image.

Lexicon Verification

A program *verify_lex* is available to check the syntax within a lexicon.

Collocation Data

A program *word_frequency* calculates the collocation of words within a given corpus. It can then generate a sorted list of the most frequently occurring pairs of words, which is useful for identifying fixed multi-word expressions.

9. SELECTION

The SPQR module (Selectional Pattern Queries and Responses)[20] is designed for two purposes: to improve the portability of PUNDIT through semi-automated acquisition of domain-specific semantic information, and to improve the accuracy and efficiency of the parser.

9.1. Overview of SPQR

SPQR operates by interactively and incrementally collecting selectional patterns, which represent information about the semantic acceptability of certain lexical co-occurrence patterns, (e.g., subject-verb-object), found in partially constructed parses. SPQR operates by analyzing the ISR of partially constructed parses, extracting syntactic patterns (either head-modifier or predicate-argument patterns) from the ISR, and finally either allowing a parse under construction to proceed, or failing the parse depending on the semantic compatibility of the components of the syntactic pattern. If SPQR encounters an unknown syntactic pattern, it can query the user about its acceptability, store the user's response in a pattern database, and thus incrementally build up a database of acceptable and anomalous patterns which it consults as it analyzes ISRs. The module has proved to be a valuable tool for porting PUNDIT to new domains and acquiring essential semantic information about the domains. Preliminary results also indicate that SPQR causes a six-fold reduction in the number of parses found, and about a 40% reduction in total parsing time. More information about this module can be found in Volume II, under *A User's Guide to the Selection Module.*

9.2. Methodology

The essential feature of our parser which facilitates the collecting of syntactic patterns is the ISR produced by the syntactic analyzer. The structure of the ISR is regular and can be analyzed to display the underlying syntactic patterns as they are generated during parsing. A typical ISR is shown in Figure 13.

SPQR is invoked by two restrictions which are called after the BNF grammar has assembled a complete NP (and constructed the ISR for that NP), and after it has assembled a complete clause (and constructed its ISR). The program operates by presenting to the user a syntactic pattern (either a head-modifier pattern or a predicate-argument pattern) found in the ISR, and querying him/her about the acceptability of that pattern. If the pattern describes a relationship that can be said to hold among domain entities (i.e., if the pattern occurs in the sublanguage), the user accepts the pattern, thereby classifying it as good. The analysis of the ISR and the parsing of the sentence are then allowed to continue. If, however, the pattern describes a relationship among domain entities that is not consistent with the user's domain knowledge or with his/her pragmatic knowledge (i.e., if the pattern cannot or does not occur in the sublanguage) the user rejects it, thereby classifying it as bad, and signalling an incorrect parse. This response causes the restriction which checks selection to fail, and as a result, the parse under construction is immediately failed, and the parser backtracks.

As the user classifies these co-occurrence patterns into *good patterns* and *bad patterns*, they are stored in a pattern database which is consulted before any query to the user is made. Thus, once a pattern has been classified as good or bad, the user is not asked to classify it again. If a pattern previously classified as bad by the user is encountered in the course of analyzing the ISR, SPQR consults the database, recognizes that the pattern is bad, and automatically fails the parse being assembled. Similarly, if a pattern previously recorded as good is encountered, SPQR will recognize that the pattern is good simply by

```
[past,repair,
   subj([tpos(the),
      [nvar([engineer,singular,_])]]),
   obj([tpos(the),
      [nvar([sac,singular,_])],
      adj([pastpart,break])])]
```

Figure 13.
ISR for the sentence *the engineer repaired the sac*

consulting the database, and allow the parsing to proceed. A suite of tools is provided to inspect and edit the patterns collected during parsing (see *The User's Guide to the Selection Module*, in Volume II).

The selectional mechanism as described so far deals only with lexical patterns (i.e., patterns involving specific lexical items appearing in the lexicon). However, we have implemented a method of generalizing these patterns by using information taken from the domain *isa* (generalization/specialization) hierarchy to construct semantic class patterns from the lexical patterns. After deciding whether a given pattern is good or bad, the user is asked if the relation described by the pattern can be generalized. In presenting this second query, SPQR shows the user all the super-concepts of each word appearing in the pattern, and asks for the most general super-concept(s), if any, for which the relation holds.

Let us take as an example the noun-noun pattern generated by the compound nominal *oil pressure*. While parsing a sentence containing this expression, the user would accept the noun-noun pattern [*oil, pressure*]. The program will then show the user in hierarchically ascending order all the generalizations for *oil* (*fluid, physical_object*, and *root_concept*), and all the generalizations for *pressure* (*scalar_quantity, object_property, abstract_object*, and again *root_concept*). The user can then identify which of those super-concepts of *oil* and *pressure* can form a semantically acceptable compound nominal. In this case, the correct generalization would be [*fluid, scalar_quantity*], because

- The fluids in the domain are oil, air, and water; the scalar quantities are pressure and temperature; and it is consistent with the domain to speak of the pressure and the temperature of oil, air, and water.
- We cannot generalize higher than *fluid* since it would be semantically anomalous to speak of *physical_object pressure* for every *physical_object* in the domain (e.g., one would not speak of *connecting_pin pressure* or *gearbox pressure*).
- We cannot generalize higher than *scalar_quantity* since *shape* is also an *object_property*, and it would be infelicitous to speak of *oil shape*.

As with the lexical-level patterns, the user's generalizations are stored for reference in evaluating patterns generated by other sentences. The obvious advantage of storing not just lexical patterns but also semantic patterns is the broader coverage of the latter: Knowing that the semantic class pattern [*fluid, pressure*] is semantically acceptable provides much more information than knowing only that the lexical pattern [*oil, pressure*] is good.

10. SEMANTICS

The semantic analysis approach used by PUNDIT has two separate, but interrelated components: 1) the algorithm that controls the execution of the rule-driven semantic analysis and 2) the theory of lexical semantics captured by the

rules themselves. The basic approach described below was originally designed for the analysis of main clauses where the PREDICATING EXPRESSION was the verb. It was an independent system that assumed the existence of separate components to parse sentences and to resolve referents of noun phrases, and performed a very rudimentary time analysis. The implementation of PUNDIT has caused the rule-driven semantic analysis approach to be fully integrated with a syntactic parser, a reference resolution component, and a sophisticated time analysis component. In addition, the analysis algorithm has been extended to cover predicating expressions in a full range of syntactic environments, including noun phrases and modifiers as well as verbs. Ports to new domains have also made demands on the theory of lexical semantics, which has been regularized to follow more traditional linguistic classifications for the semantic roles, and has been extended to cover a much broader range of verb subcategorizations.

Semantic analysis in PUNDIT is based on Inference Driven Semantic Analysis [23] which decomposes verbs into component meanings and fills their semantic roles with syntactic constituents. The result of the semantic analysis is a set of PARTIALLY instantiated semantic predicates which is similar to a frame representation. To produce this representation, the semantic components share access to the DOMAIN MODEL that contains generic descriptions of the domain elements corresponding to the lexical items. The model includes a detailed representation of the types of assemblies that these elements can occur in. The semantic components are designed to work independently of the particular model by relying on an well-defined interface.

In order to produce the correct semantic representation, the predicating expression is first decomposed into a semantic predicate representation appropriate for the domain. The arguments to the predicates constitute the SEMANTIC ROLES of the predicating expression, which are similar to verb cases. Semantic roles can be filled either by a syntactic constituent supplied by a mapping rule or by reference resolution, requiring close co-operation between semantics and pragmatics. A proposed role filler must satisfy the semantic class restrictions on the role. In order to control how roles are filled, certain semantic roles are categorized as OBLIGATORY, indicating that they must be filled by a syntactic constituent. Other roles, in the context of particular verbs, are categorized as ESSENTIAL, which signals pragmatics to fill the role even if there is no syntactic constituent available. The default categorization is NON-ESSENTIAL, meaning the role does not need to be filled. These classifications are explained in more detail with extensive examples in [24] and [6]. Details of the clause analysis algorithm, as well as differences required for the analysis of other predicating expressions are described in[7].

10.1. Predicating Expressions

The original clause analysis algorithm has been extended to cover predicating expressions in a full range of syntactic environments, including noun phrases and modifiers as well as verbs. One implementation of the algorithm is used to process all of these types of predicating expressions. This is done by having the interpreter operate in a different mode for each different type of syntactic environment.

Since each syntactic environment containing a predicating expression can have embedded within it another syntactic environment containing another predicating expression, the semantic analysis algorithm must be recursive in the same way that the syntactic parser is. For example, in the analysis of *Investigation of decreased pressure revealed metal contamination in oil*, the initial call to semantic analysis pertains to the analysis of the *reveal* clause. This in turn requires the analysis of the *investigation* phrase, a nominalization, which is the proposed filler for one of the semantic roles of *reveal*. The analysis of *investigation* requires the analysis of the noun predicate *pressure*, as the head noun, and a proposed filler for one of *investigation*'s semantic roles. The modifier of *pressure* is *decreased*, the participial form of the verb *decrease* which is being used as an adjective. This is also a predicating expression which has to be analyzed. After successfully completing the analysis of each of these phrases, the interpreter will proceed to analyze the other nominalization, *contamination*, the proposed filler for the second semantic role of *reveal*, and another predicating expression, and so on. In all of these levels of recursion, the relevant time information and current discourse context must be kept straight so that the final representation is accurate. This requires a carefully integrated control structure for the semantic and pragmatic components. The tight integration of control has been particularly effective in the achievement of two of the original research goals: 1) the filling in of implicit information, and 2) the recovery of information from incomplete sentences. The interactions with reference resolution and the time component are discussed in detail in[24, 21, 6].

10.2. Adapting the Analysis Process to New Predicating Expressions

Predicating expressions occur in a full range of syntactic environments in the domains we have analyzed, and each receives a different representation from the ISR. For each of these predicating expressions, adjustments must be made to the semantic and pragmatic components to insure the production of an accurate final representation for the IDR. This representation must include the instantiated semantic decomposition, the correct resolution of referents, and an accurate analysis of intra-sentential time relations. In practice, there are many types of predicating expressions that do not behave semantically as either simple clauses or simple noun phrases. There are verbs that can be used as nouns, e.g., nominalizations such as *failure* and *monitoring*, and nouns that can be used as verbs, e.g., deverbal nouns such as *crack*. Verb participles can be used as

noun modifiers, and nouns can be used to modify other nouns. Each new type of mixed-category predicating expression requires a customized version of the semantic analysis algorithm that makes allowances for its particular requirements. For instance, nominalizations and non-finite subordinate clauses need to go through time analysis, even though they do not have syntactically marked tense. By allowing the tense of the matrix clause to be passed around as a parameter during the recursive calls to semantic analysis, time analysis can be given the information it needs to produce an accurate final representation.

Figure 14 summarizes the different types of predicating expressions that the system handles, for at least some cases. A detailed discussion of the operation of the interpreter for the different types of predicating expressions can be found in [7]. In the last two years of this project, we have added polysemous verbs, polysemous noun phrases, relative clauses, and adverbial modifiers to this list. The remaining subsections of this section describe the semantics coverage in greater detail.

10.3. Conjunction

PUNDIT semantics currently handles the connectives *and* and *or*. Since exactly the same processing is provided for both, the following discussion will be in terms of *and*. Phrases of almost every category may be conjoined: adverbs (*quickly and efficiently*), nouns (*nuts and bolts*), verbs (*They aimed and fired*), clauses, etc. The cases which PUNDIT semantics is equipped to handle, as far as conjunction is concerned, are conjoined sentences or clauses, and conjoined noun phrases.

Conjoined Sentences

Example: (1-27) DEPARTED STATION AND RETURNED TO CV.

PUNDIT analyzes and produces a representation of the meaning of each sentence, as though the sentences had occurred independently (i.e., *Departed station. Returned to CV*).

Conjoined Embedded Clauses

Example: MILLER REPORTS THAT ATTACK WAS SUCCESSFUL AND SUB IS OOA.

Clauses which occur as subjects or objects are analyzed syntactically as noun phrases, and are handled by noun phrase semantics (see below). In this example, the analysis would be that Miller reported two situations.

Conjoined Noun Phrases

For conjoined noun phrases (and plurals), three basic readings exist:

The collective reading:

Verb Phrases
- Main clause verbs
- Subordinate clause verbs
- Non-finite subordinate verbs
- Conjoined verbs
- Polysemous verbs

Noun Phrases
- Nominalizations
- Noun predicates
- Conjoined noun phrases
- Polysemous noun phrases

Prepositional Phrases
- Prepositional phrases attached to nouns
- Prepositional phrases attached to verbs
- Prepositional phrases attached to assertions

Adjectival Modifiers
- Adjectives
- Nouns that are left modifiers
- Participles that are left modifiers
- Relative clauses

Adverbial Modifiers
- Time adverbs modifying assertions
- Goal adverbs modifying assertions
- Other adverbs modifying assertions, such as manner adverbs

Figure 14.
The Range of Syntactic Environments for Predicating Expressions

Example: (2-25) ORIG AND FANNING FORMED SAU.
(together, the message originator and the Fanning formed a surface action unit)

The distributive reading:

Example: (5-7) MILLER AND WAINWRIGHT ENGAGED KOBCHIK WITH MISSILES.
(Miller engaged Kobchik with missiles and Wainwright engaged Kobchik with missiles. Two separate engagements; different missiles)

The reciprocal reading, with 'symmetric predicates'

Examples:
(5-8) BOTH FRIGATES USING HULL NUMBER 625. IDENTITY OF BARSUK AND KOBCHIK MAY BE REVERSED.
(identity of B. may be reversed with the identity of K.)

MILLER AND BARSUK EXCHANGED FIRE.
(Miller fired on Barsuk and Barsuk fired on Miller)

PUNDIT noun phrase semantics interprets all conjunction (and plurals) as collective. For this reading, noun phrase semantics analyzes each element of the conjunction in turn. After processing each noun phrase, it creates a 'group' of entities, adds up the cardinality of each member to determine the number of objects in the group, and determines the semantic class of the group based on the class of each member. In the example below, a group is created whose members are the Miller and the Wainwright; the class of the group is *US_platform*, and its cardinality is two.

Example: (5-7) MILLER AND WAINWRIGHT ENGAGED KOBCHIK WITH MISSILES.

10.4. Noun Phrases and Prepositional Phrases

PUNDIT noun phrase semantics is responsible for analyzing and representing the meaning of noun phrases (NPs) and prepositional phrases (PPs). It is important to know that NP semantics does not automatically analyze every NP and PP in a sentence, but is rather under the control of the Semantic Interpreter. If, in the course of processing a sentence, the Semantic Interpreter finds a constituent that maps to a thematic role in the semantic representation which it is building, then it calls a procedure to analyze that constituent. If the constituent is not a clause or an adjective, then NP semantics is invoked.

If the constituent is a prepositional phrase, NP semantics analyzes its object (e.g. in (6-10) *TAKING SWIMMER PICKUP BOAT UNDERFIRE WITH SMALL ARMS*, *small arms* is the object of the PP *with small arms*).

If the constituent is an NP, noun phrase semantics analyzes the constituents of the noun phrase as described below, and creates a list of predications which represent the meaning of the noun phrase.

Readings of Noun Phrases

A distinction is often made between referential and non-referential readings of NPs. The distinction is illustrated by the two sentences below:

A carrier was sighted.
The USS Kitty Hawk is a carrier.

In the first sentence, *a carrier* describes a particular individual (the referential reading); in the second sentence, *carrier* is interpreted as a property of the USS Kitty Hawk (the non-referential reading). The distinction is an important one for reference resolution, since individuals and properties, once introduced, are then referred to differently. Compare:

A carrier was sighted. It attacked.
The Kitty Hawk is a carrier. The Constellation is one also.

All indefinite NPs which occur as complements of the verb *be* are analyzed by NP semantics as properties, and their meaning is represented as *predicative(Head)*, where *Head* is the head noun (e.g. *predicative(carrier)*). All other NPs are analyzed as referential, and they evoke discourse entities.

Analysis of Referential NPs

Noun phrases in general consist of an article, a head noun, and (optionally) a number of modifiers of various types (nouns, adjectives, relative clauses, appositives, etc.). NP semantics must determine how each of the constituents of the NP contributes to the meaning of the NP as a whole. The major cases are as follows:

Nominalizations:

Nominalizations are NPs which are related to verbs, and which have a semantics similar to that of clauses.

Example: (5-9) TEXAS CONDUCTED UNSUCCESSFUL MISSILE ATTACK AGAINST ADM GOLOVKO

Sentence paraphrase: USS Texas attacked unsuccessfully with missile(s) against the Admiral Golovko.

If the head noun of the NP is defined in PUNDIT's semantic rules as a nominalization, NP semantics simply calls the Semantic Interpreter to do the analysis. In this case, *attack* would be defined as a nominalization. The semantic rules for *attack* would be used to build up the semantic representation of *missile attack against Adm Golovko*, and the semantic rules for the modifier *unsuccessful* would be used to complete the representation. The result would be semantic representations for two situations: the attack, and the state of non-success of the attack.

Noun Predicates:

Noun predicates are nouns which are not nominalizations, but which have an argument structure. The analysis of such nouns and their arguments is, like nominalizations, governed by semantics rules.

Example: (2-50) FIRED 30 ROUNDS OF AAC AND 2 MISSILES ON BARSUK RESULTS OF ATTACK UNKNOWN AT THIS TIME.

Since *results* are always results *of* something, it is useful to analyze *result* as a semantic predicate which takes an argument. If there is a semantics rule defined for the head noun, as there would be in this case, NP semantics calls the Semantic Interpreter to do the analysis. The result would be a representation of the predicate-argument structure of *results of attack.*

Ordinary NPs:

For ordinary noun phrases (not nominalizations or noun predicates), NP semantics analyzes the head noun and its modifiers, and creates representations which are intended to capture the semantic contribution of each. In general, processing the head noun will result in the creation of a discourse entity and an 'id' relation (singular NPs) or an 'is_group' relation (plural NPs), and the modifiers will then be related to the head noun.

The analysis of modifiers is complex, and PUNDIT provides only a partial treatment of this difficult area. Only highlights can be given here.

Articles: the article, if present, is considered to signal whether the NP is definite or indefinite. This information is used by Reference Resolution, and does not appear in the semantic representation of the NP.

Adjectival modifiers: if the modifier has a semantics rule associated with it, the Semantic Interpreter is called to build a semantic representation for the modifier and its argument (the head noun). If there is no semantics rule, then the domain model is examined for a specific relationship between the modifier and the head noun. If none can be found, NP semantics simply creates a representation of the form *unspecified_relationship(Modifier,Head).* Each of these cases will give rise to different types of semantic representation.

These representations can be illustrated with three modifiers:

an **unknown** submarine
a **nuclear** submarine
a **new** submarine

For the first example, suppose that there is a semantics rule for the modifier *unknown*. Then the analysis of *unknown ship* will result in a representation of a state of 'not knowing' involving some experiencer and a ship. For the second example, suppose that in the domain model, submarines and other platforms have a *power_source* attribute, and that 'nuclear' is a possible power source. The analysis of *nuclear submarine* would result in a representation of the form *power_source(nuclear,[submarine1])*. For the third example, suppose that there is no semantics for *new*, and that there is nothing in the domain model that relates *new* to *submarine*. In this case, NP semantics simply creates the predicate-argument structure *unspecified_relationship(new,[submarine1])*.

Noun modifiers: the interpretation of noun-noun compounds is heavily dependent on domain knowledge. For example, in *terrier missiles*, the noun modifier specifies the type of missile, while in *Miller missiles*, the noun modifier specifies the possessor. Possible relationships are defined in the domain model for each domain. If no relationship can be established, NP semantics creates an *unspecified_relationship* relation.

Appositives: determining the relationship between an appositive (or parenthetical) and its head is arguably a task for pragmatics rather than semantics, but NP semantics attempts to discover a semantic relationship between the two, using the domain model. Two types of relation are supported, as illustrated below.

Pronouns and other reduced forms:

Most of the work of interpreting pronouns is done in reference resolution, since semantically, these expressions are nearly empty. The task of the noun phrase semantics component with respect to pronouns is therefore mainly to prepare appropriate representations for input to pragmatics. The structures handled are: wh-gaps, as in *Which ships attacked?*, lexical pronouns (*I,you,it...*), *one*-anaphora, as in *a new one from supply*, elision, as in *Replaced sac*, and relative clause gaps, such as the object of *attack* in *the ships that we attacked.*

Identity of Reference:

Example: (5-16) INTEND ... ATTACK SECOND KYNDA (ADM GOLOVKO)

The Admiral Golovko is a Kynda-class cruiser, and the appositive clarifies the reference of *second Kynda*. In the semantic representation for *second Kynda*, NP semantics will substitute *Golovko* for *Kynda*.

Specification of a Role Filler:

This case does not occur in the RAINFORMs corpus, but does occur in the corpus of Trident Trouble and Failure Reports, as in:

Example: REPLACED INTERLOCK SWITCH (SN 8026).

Switches have serial numbers, and in this case the appositive specifies the value of the serial number for the interlock switch.

10.5. The Semantics Rule Editor

An important addition to our set of tools is the Semantic Rule Editor. The Semantic Rule Editor allows the entering and subsequent editing of a word's semantic information through one simple interface. The relevant information for a given word is presented in a format patterned after a paper form. Figure 15 depicts the form's layout for the verb *attack*.

The contents of each of the fields on the form is strictly controlled by the SRE. For example, when entering a verb's noun-phrase mapping constraints, a list of

---

```
Word: attack

decomposition:
 >  attackP( actor theme instrument )

clause mapping rules:
 > actor:  subj pp(by)
 > theme:  pp(against) pp(on) obj
 > instrument:  pp(with)

noun phrase mapping rules:
 > actor:  noun pp(of)
 > theme:  pp(on) pp(against)
 > instrument:  pp(with) noun

constraints:
 > actor:  [ class(platform_property_C) ]
 > theme:  [ *none* ]
 > instrument:  [ class(projectile_C) ]
```

The items in bold face represent editable field values on the form.

Figure 15.
Sample Entry in the SRE

---

valid grammatical roles is presented to the user to select from; no other values may be entered.

Along with managing the editing and creation of semantic information, the SRE interacts with PUNDIT to include the new data in the running image and to store all the semantic information in a permanent file.

11. TIME

One of PUNDIT's unique features is its sophisticated component for temporal analysis[25, 26]. Temporal information is highly context dependent and distributed among many interdependent sentence elements. The interpretation of present tense, for example, varies in each of the following sentences, depending on other elements in the sentence such as the type of adverbial, the number and definiteness of the subject, and lexical aspect (i.e., inherent temporal content of individual verbs, adjectives and nouns).

The submarine is ooa.
Submarines are subsurface platforms.
The Constellation refuels on Fridays.
The Nimitz leaves tomorrow.

The task of temporal interpretation depends on determining what entities and situations that have been referred to have time associated with them, relating to them the appropriate components of time (e.g., durations, onsets, terminations), and representing how times are ordered with respect to one another. The following is a list of some of the relevant parameters that have to be taken into account in temporal interpretation.

sentence type (assertion, imperative, question)
intra-sentential temporal relations
tense
perfect and progressive
lexical aspect
adverbial expressions
other lexical information
other issues: modality, inter-sentential time

The following subsections review each item in this list, discussing how the relevant temporal information is represented.

11.1. Sentence Type

All full declarative and interrogative sentences have a tensed matrix verb or auxiliary verb, or have a modal auxiliary. Imperatives have a bare verbal form with no tense. PUNDIT's temporal component is geared to interpreting the declarative sentences found in message text, although we have now extended it to handle questions and imperatives as well. The following sections indicate how temporal information in declarative sentences is processed.

11.2. Intra-sentential Temporal Relations

A single sentence may contain more than one clause. The possibilities include:

Clausal Adjuncts

Clauses introduced by subordinating conjunctions (e.g., *because, when*) are adjuncts. If the subordinating conjunction is a temporal adverb (e.g., *when*), the temporal relation between the matrix and adjunct clauses is computed.

Embedded Clauses

Tensed embedded clauses are interpreted like independent sentences. E.G., for *The golovko believed that the constellation went sinker*, the matrix clause with *believe* gives rise to a past **belief** state and a past **go sinker** event, but the two past times are not ordered with respect to one another. Untensed clauses are interpreted as if they had the tense of the matrix clause; e.g., *The golovko appeared to be burning.* Again, the times of the two clauses are not ordered with respect to one another.

11.3. Tense

The present and past tenses each have a very simple rule of interpretation. A set of interpretive procedures decides whether these rules can be applied in any given case, and what situations they apply to. The past tense locates a time prior to the discourse time by a binary **precedes** relation. The present tense locates a time as coinciding with the discourse time by a binary **coincide** relation. Currently, the discourse time is simply represented as the constant **discourse_time**, a temporal index for the time at which an utterance occurs, or when a text is produced.

Each situation is assigned a temporal structure, one component of which is a temporal index referred to as the **event time**. Three temporal indices are used in interpreting all the simple and compound tenses. For the simple tenses, the third index, the **reference time**, is always identical to the **event time**. A distinct **reference time** is required for interpretation of the past perfect. The simple tenses always locate an **event time** with respect to the **discourse_time**. Since an **event time** is always represented as a unary **moment** relation, whose argument is the referential index for the situation itself, the temporal relations computed by tense are always of the form

precedes(moment([EVENT1]),discourse_time)

or

coincides(moment([EVENT1]),discourse_time)

11.4. Perfect and Progressive

Interpretation of all the compound tenses (present and past perfect, present and past progressive, present and past perfect progressive) is supported. The following rules summarize the treatment of the compound tenses.

If a verb phrase is in the perfect, the situation it refers to is in the past. If present perfect, the **reference time** is identical to the **discourse time**. If past perfect, the **reference time** is distinct from both of the other temporal indices; it follows the **event time** but precedes the **discourse time**. The reference time is used in interpreting temporal adverbs, thus given the sentence *the golovko had attacked when the submarine went sinker*, the **attack** event would be represented as preceding its reference time (**moment([ref_moment])**); the **go_sinker** event would be represented as coinciding with that reference time, or in other words, the **attack** event precedes the **go_sinker** event:

precedes(moment([attack1]),moment([ref_moment1])

coincide(moment([ref_moment1]),moment([sinker1])

Inferences are often associated with the perfect, for example, that the state resulting from an event persists up through the present (e.g., *the golovko has disappeared*). Also, the perfect can have pragmatically conditioned interpretations which are not temporally relevant. No inferences associated with the perfect tenses are currently supported.

Progressive affects the interpretation of temporal structure. Briefly, the **event time** derived for a progressive verb is always inside some duration associated with a situation. The consequence of this rule is that a verb which implies a culmination point is interpreted differently if used in the progressive. Take for example, a change of location verb like *return*. The following sentence would give rise to the inference that the golovko is at the station:

the golovko returned to station

A corresponding sentence in the progressive would not give rise to a culmination point, thus blocking the inference that the golovko is at the station:

the golovko is returning to station.

11.5. Lexical Aspect

Verbs are recognized as having lexical aspect. Nouns that are nominalizations also are recognized as having aspect. Three categories of lexical aspect in verbs are supported: state, process, and transition event verbs.

State

A state holds over an indefinitely long period of time with no change; e.g., cognitive state verbs *know, believe* and all predicate adjectives, *be unknown, be successful.* Some passive participles can be interpreted adjectivally, particularly in the present tense, e.g., *The submarine is submerged/lost/damaged.*

Process

A process also holds over an indefinitely long period of time. It differs from a state in that the participants in the process undergo change; e.g.,

approach, move, hover.

Transition event

A transition event is a change-of-state or change-of-location, i.e., an event which culminates in a new situation. The culmination point, or transition, is represented as a moment associated with the event itself. This moment is represented as the onset of a period of time associated with the resulting situation. Examples include *join, break, expend.*

There are no special aspectual categories for verbs of gradual change (e.g., *increase, burn*), punctual verbs (e.g., *cease, recognize, disappear*), or for cyclic verbs (e.g., *fluctuate*). They are classified as transition events or processes.

A clause with a tensed verb may refer to a specific situation, or to a kind of situation that tends to recur, e.g., *the Constellation refueled (yesterday)* (specific) versus *the Constellation refuels (on Fridays)* (general kind). PUNDIT only represents specific situations and the actual times at which they have been asserted to occur. Events and processes in the simple present tense are generally interpreted as referring to types of situations, thus are not represented.

11.6. Adverbial Expressions

Lexical items, phrases and clauses with adverbial content are syntactically and semantically very heterogeneous. Temporal adverbs that are prepositions or subordinating conjunctions are supported. Currently, this set includes *after, before, during, while, when.* Types of adverbs which aren't supported include

Noun Phrase adverbs. EG., *Monday, this week*
Durational adverbs. EG., *for five hours*
Frequency adverbs. EG., *often, frequently*
Temporal quantifiers. E.g., *whenever, every time that . . .*

12. PRAGMATICS

The respective domains of Semantics and Pragmatics are not clearly demarcated in current linguistic theory, and the division of labor continues to be a controversial area. The point of view taken in PUNDIT is that Semantics is responsible for supplying the literal meaning (or meanings) of a sentence in isolation, based on the meanings of its parts. Pragmatics must supply *the conveyed meaning of utterances in context.* It begins where Semantics leaves off: given literal meaning, linguistic form, and a context, it must explicate the meanings that are conveyed in that context.

As a simple illustration of the different tasks, consider the utterance of *I want to see all ships which are in the Indian Ocean today.* The literal meaning of this sentence might be glossed as the speaker being in a state of desiring to be in a state of visually experiencing some set of ships. A theory of semantics would then give us the set of conditions under which this is true or false. But clearly we cannot interpret this without reference to the context: Pragmatics must

explain how such expressions as *I*, *today* are interpreted, and must furthermore explain how such utterances are used to convey requests, and why it is that *Yes* or *That's interesting* are not appropriate responses.

Having distinguished Pragmatics from Semantics, we must be careful to further distinguish Pragmatics from the more general study of cognition. It is not the task of Pragmatics to explain how humans reason (in general), or how humans model the world (in general). More specifically, in terms of computational linguistics, it is not for Pragmatics to provide a general reasoning capability or to furnish a knowledge base and the procedures which operate on it. These are general cognitive facilities which go beyond linguistic competence, although it is clear that such facilities must be available to a system which is modelling language understanding.

12.1. Reference Resolution

A fundamental feature of all human languages is the facility for reference, and this facility is mediated by models of the world, of beliefs, and of the discourse. Language is symbolic: if we wish to convey some information about a ship using language, we cannot include the physical ship itself in our utterances. Instead, we must use the linguistic resources at our command in such a way that the hearer (or reader) can recognize our intention, and in such a way that s/he can successfully pick out the intended entity.

PUNDIT's Reference Resolution component [5, 4] models aspects of the recognition competence of language users: specifically, the ability of a hearer or reader to recognize the referential intentions of a speaker or writer, and to pick out the intended entity on the basis of some linguistic expression such as *USS Constellation*, *the carrier*, or *it*.

A brief review of the process of semantic analysis will serve to elucidate the relationship between Reference Resolution and the Semantic Interpreter. When the Interpreter is processing a sentence such as (2-4) *FIRED ONE HARPOON AT UNKNOWN CONTACT*, it creates a predicate-argument structure based on the verb (*fire*), and then tries to fill the argument positions with constituents from the sentence. The constituents in this case are the invisible subject (which is considered a null pronoun), the object *one harpoon*, and the prepositional phrase *at unknown contact*. After the Interpreter maps a constituent to an argument position in the semantic representation, it analyzes the meaning of the constituent, and then calls Reference Resolution to determine what the linguistic expression refers to.

Reference Resolution proposes candidates from the context, until one is found which satisfies the semantic constraints on the argument of the verb. For example, the null subject pronoun has no semantic content and could potentially refer to anything in the context; in this case, the constraint that only platforms can fire weapons is crucial to finding the right referent. By contrast, *one harpoon* has sufficient semantic content to allow Reference Resolution to

recognize that there is no such entity in the context. It will therefore create a new entity, and will recognize that it is a missile which is associated with a previously mentioned ship.

At that point, the reference is considered resolved, and the Semantic Interpreter goes on to fill the next argument position. When the semantic representation has been completed, Reference Resolution is called upon again to assign a discourse entity to the result; this is because sentences evoke situations, which can be referred to later: *FIRED ONE HARPOON... RESULTS* (of that action) *UNKNOWN.*

Reference Resolution incorporates knowledge about how linguistic devices such as articles and pronouns are used to signal referential intentions. It uses such linguistic cues, when they are present, to determine whether to create a new entity or whether to search the context for an 'old' entity.

In searching for a previously mentioned entity, Reference Resolution is guided by the representation of the meaning of the referring expression. Thus, for example, in trying to resolve the reference of *the submarine*, Reference Resolution looks for a previously mentioned entity which is a submarine; in resolving the reference of *the nuclear sub*, it will look for an entity which has the property of being both a sub and having a nuclear power source.

When the referring expression is a pronoun (other than *I* or *you*), there is insufficient semantic content for Reference Resolution to find the intended entity on the basis of meaning alone. The most common pronoun in messages is in fact the null pronoun, where there is no expression of the pronoun at all, and hence no semantic content, e.g. (2-1) *REGAINED CONTACT.* There is also no semantic content (or linguistic expression) in the case of *implicit arguments.* For example, in nominalizations such as (2-22) *ATTACK WITH ASROC AND TORPEDOS* one or more arguments are implicit: here, the attacker and the attackee. Reference Resolution may be called upon to propose candidate entities from the context to fill these arguments. In all of these cases, there is little or no semantic content to guide Reference Resolution in its search for an appropriate entity. To handle these cases, the system incorporates an algorithm which orders discourse entities by saliency and produces a *focus list*; it is the focus list which governs the search for the referents of pronouns and implicit arguments.

Reference Resolution is also able to handle reference to entities which have not been explicitly introduced in previous discourse, but whose presence is inferrable from the context. For example, in the following text, *the nose gear* has not been previously mentioned, but it is inferrable from the mention of the A-3 aircraft.

(6-39) SELECTED MISHAPS: A. FOLLOWING AN APPROACH TO A CV, AN A-3 ATTEMPTED A WAVE-OFF BUT ENGAGED NUMBER FOUR GROSS DECK PENDANT WHILE AT AN ALTITUDE OF ABOUT 3 FEET, COLLAPSING THE NOSE GEAR ON IMPACT WITH THE DECK ...

For the OPREPs domain, PUNDIT's coverage of reference was extended in several important ways. The most interesting new functionality is the ability to handle some cases of forward anaphora, where the anaphor (pronoun or elision) precedes the full noun phrase antecedent. We are not aware of any previously reported treatments of this phenomenon in other natural language systems. Processing of forward anaphora was achieved by making reference resolution more sensitive to the syntactic context of the references it is processing. Previously, elided subjects in sentence modifiers had been treated the same way that all elided elements are treated---as pronouns with a preference for referents in focus (that is, introduced in earlier sentences or globally available). In fact, however, elided subjects in sentence modifiers show a preference for the surface subject of the main clause as referents. Reference Resolution now uses this information to correctly interpret these elided subjects. This can be seen in the following OPREP example of forward anaphora, where PUNDIT correctly interprets the subject of *direct* as USS Merrill and Sterett:

AS DIRECTED BY CDS1 USS MERRILL AND USS STERETT EXECUTED COORDINATED 2 HARPOON SALVO ON KRIVAK II

12.2. The IDR

PUNDIT's semantic and pragmatic components take the ISR as input and produce a final representation of the information conveyed by the sentence which includes a decomposition of verbs into a structure of more basic predications, resolution of anaphoric references, and an analysis of temporal relations. The resulting data structure is known as the Integrated Discourse Representation, or IDR.

The IDR will be illustrated for the following sentence:

> *Visual sighting of periscope followed by attack with asroc and torpedos.*
> *Translation: The visual sighting of a periscope*
> *was followed by an attack (on the submarine) with anti-submarine rockets and torpedos.*

This particular sentence is characteristic of the sort of input PUNDIT has been designed to handle. Note the ellipsis typical of message sublanguages.

The IDR for the example sentence is shown in Figure 16. Its major segments are labelled *Ids*, *Properties*, *Events and Processes*, *States*, and *Important Time Relations*. The *Ids* segment lists all the *id*, *is_group*, and *generic* predications derived during the analysis of the example sentence. *Generic* relations are established primarily to support subsequent reference through generic *they* or *one*-anaphora. *Id* relations indicate the semantic type of each non-group discourse entity, while the *is_group* relations specify the semantic type, members, and cardinality of each group-level discourse entity. Thus for example the *id* relation for the entity *[sight1]* derived from the nominalization *visual sighting of periscope*, indicates that the entity is an *event*, while the *is_group* relation for the

entity *[projectiles1]* indicates that the entity is a group of projectiles, consisting of an unknown number of rockets and torpedos. (Labels for discourse entities are derived from the lexical head of the expression and are typically enclosed in brackets. These labels are arbitrary; *[entity2]* would do equally well.)

Relations in the *Properties* segment of the IDR are heterogeneous: these are miscellaneous relations derived in the course of processing noun phrases. Prenominal adjectives typically give rise to such relations; processing of noun-noun compounds may generate *unspecified_relationship* predications if no relationship between the nouns can be derived from domain knowledge. In the current example, the *reportingPlatform* relations are generated by a procedure which creates a default entity if the identity of the message originator is not known.

The *Events and Processes* and *States* segments of the IDR contain predications over discourse entities which denote situations; see[25] for a more detailed discussion of the semantics of situations. Typically it is the processing of a clause or a nominalization which gives rise to a situation entity, and if the situation is an event, then an entity will be generated for the resulting state as well. The main predicate is the type of situation (event, state, or process), and each predication has three arguments:

The discourse entity;
The associated semantic representation;
A moment or period of time for which the situation holds.

For example, the first predication in the **Events and Processes** segment in Figure 16 was derived from processing the ISR for the nominalization *visual sighting of periscope*. This particular predication asserts that the referent introduced by the gerund *sighting* denotes an event; the semantic representation was constructed based on the semantics rules for the verb *sight*. In the second argument of the predication, the *becomeP* operator takes as its argument the semantic representation that gives rise to a new state that is entered into, *[sight2]*. The third argument of the predication, *moment([sight1])*, should be interpreted functionally as returning the moment at which the transition into the state in question occurred. Information about this new state, *[sight2]*, is provided by a predication in the *States* field.

The final segment of the IDR lists the temporal relations which were analyzed as holding among the situations. Note in particular that since the verb *follow* is defined as a temporal operator, PUNDIT has correctly established the temporal relationship between the sighting and the attack.

13. KNOWLEDGE REPRESENTATION AND REASONING

Pundit has the capability to interact with any frame-based knowledge representation system. M-Pack, the knowledge representation system that is currently being used with Pundit, is a version of KNET that has been enhanced to accommodate multiple inheritance[11, 22].

```
Ids:
generic(torpedo)
is_group([torpedos1],members(torpedo,[torpedos1]),numb(_A))
generic(anti^submarine^rocket)
id(anti^submarine^rocket,[rocket1])
is_group([projectiles1],members(projectile,[[rocket1],[torpedos1]]),numb(_B))
id(us_platform,[us_platform1])
id(process,[attack1])
generic(periscope)
id(periscope,[periscope1])
id(us_platform,[us_platform3])
id(state,[sight2])
id(event,[sight1])

Properties:
reportingPlatform([us_platform1])
reportingPlatform([us_platform3])

Events and Processes:
event(
  [sight1]
  becomeP(sightP(experiencer([us_platform3]),theme([periscope1]),instrument(visual))
    sighted_atP(theme([periscope1]),location(_C))
  moment([sight1]))

process(
  [attack1]
  doP(attackP(actor([us_platform1]),theme(_19607),instrument([projectiles1])))
  period([attack1]))

States:
state(
  [sight2]
  sightP(experiencer([us_platform3]),theme([periscope1]),instrument(visual))
    sighted_atP(theme([periscope1]),location(_D))
  period([sight2]))

Important Time Relations:
the sight state ([sight2]) started with the sight event ([sight1])
the sight event ([sight1]) preceded the arbitrary event time (moment([attack1]))
of the attack process ([attack1])
```

Figure 16.
IDR
Visual sighting of periscope followed by attack with asroc and torpedos.

13.1. The Knowledge Base

PUNDIT uses the KNET knowledge representation formalism to store information about object subclasses and superclasses, part-whole relationships, and attributes of objects. This information can be used to help establish the semantic relationships among sentence components.

KNET is a semantic net representation based on KL-ONE[3]. The fundamental data value in KNET is the *concept*, which represents the abstract notion of an object or event. All concepts participate in an *isa* hierarchy. Other relationships among concepts are represented by *roles* which associate pairs of concepts. *Individuals* represent specific instances of concepts.

Extensions to the KNET representation have been made in order to support multiple inheritance, which allows a concept to inherit attributes from more than one of its parents in the isa hierarchy. In addition, KNET has been extended to accommodate user-defined attributes of objects, thus greatly increasing the flexibility of the representation.

The PUNDIT system has access to the KNET knowledge base via the M-PACK procedure library. Using M-PACK, PUNDIT has the ability not only to examine the the KNET knowledge base, but also to alter and extend it dynamically. M-PACK provides extensive error checking, so that the knowledge base is guaranteed never to enter an inconsistent state.

The knowledge expert may create and edit a KNET knowledge base by using the *miniBrowser* program. The miniBrowser is a text-oriented knowledge-base editor that uses the M-PACK routines, so that full consistency checking is performed.

In addition, an ASCII text representation has been developed for KNET knowledge bases. This representation has been designed to capture the most important and most commonly used features of KNET and to be relatively intuitive and syntactically simple. Using this representation, the knowledge engineer may produce a relatively compact listing of the state of the knowledge base and may use any available text editor to make otherwise difficult corrections to the knowledge base. As in the case of the miniBrowser, M-PACK routines are used to guarantee the integrity of the resulting knowledge base.

13.2. PUNDIT Interface to M-Pack

In this section, two data structures that must be utilized to establish mappings to concepts in an associated knowledge base are first described. Next, two sets of procedures that have been defined to help achieve an appropriate degree of portability are introduced.

Data structures used to establish mappings to concepts

One of the most basic data structures used in Pundit to access information within a knowledge base is called a *denotes* clause. A *denotes* clause is a two-

place predicate that establishes a mapping between the root form of a lexical item and the name of a concept within a knowledge base. From a software development point of view, it is desirable to insure that the names of lexical items and the names of the concepts they map onto are distinct, but this is not a restriction that Pundit enforces. The example *denotes* clause below specifies a mapping between the common noun *tape* and a concept named *tape_C* in some knowledge base.

denotes(tape, tape_C).

There is no reason that the lexical item *tape* couldn't be associated with more than one concept. For example, it might be associated with more than one type of physical object—magnetic tapes and adhesive tapes. It might also be associated with a type of relation in which one entity is involved in the activity of taping some other entity. To associate a lexical item with more than one concept, additional *denotes* clauses must be asserted.

Pundit currently expects all nouns, adjectives, and adverbs to be associated with concepts. It will soon expect all prepositions and verbs to be associated with concepts as well. If these mappings are not established, the system will fail to interpret messages properly.

A second type of basic data structure used in Pundit to access information within a knowledge base is called a *pkr_hook* clause. A *pkr_hook* clause has the same sort of function as a *denotes* clause, only the thing being mapped onto a concept name is not a lexical item. There is usually no way to tell by visual inspection that the first argument in a *pkr_hook* clause is not a lexical item—this is something determined by usage. There is, for example, both a *denotes* clause and a *pkr_hook* clause in the Trident domain whose first argument is *part*. In the case of the *denotes* clause, *part* is a lexical item, and in the case of the *pkr_hook* clause, it is a Prolog term used to access a concept within a Prolog clause defining a semantic constraint—the concept accessed needn't be the same one that the *denotes* clause picks out, although they do happen to be the same in this case. The *pkr_hook* clause involving the Prolog term *precedes* is used during temporal analysis and the construction of database relations to access a concept representing a certain type of temporal relation. Similarly, the *pkr_hook* involving the Prolog term *trident_default_associate* is used to identify the concept whose instances are the sorts of things that are usually what other things are a part of in the Trident domain. Most messages in this domain describe problems with components of magnetic tape units, and so it is assumed that a part belongs to some magnetic tape unit, unless there is evidence to the contrary.

```
pkr_hook(part, mechanical_device_C).
pkr_hook(precedes, precede_C).
pkr_hook(trident_default_associate, magnetic_tape_unit_C).
```

In Pundit development activities, it is important to keep in mind the need use *denotes* clauses and *pkr_hook* clauses to access concepts rather than refe ring to them directly. If this convention is violated, then the ability to swite to a different knowledge base will be compromised.

PKR compatibility procedures and PKR operations

In addition to the use of *denotes* clauses and *pkr_hook* clauses, two distinct sets of procedures have been defined in Pundit to achieve an appropriate degree of portability. The first set of procedures, called *compatibility procedures*, provide branching points at which methods for accessing information may differ, depending on whether a frame-based representation is being used or not. These procedures were introduced to avoid having to immediately replace older domain models that are not expressed in a frame-based representation.

The second set of procedures, called *PKR operations*, provides a layer of communication between Pundit and whatever frame-based representation system is being used. However, it is assumed that the representation system *is* frame-based. These operations are intended to free developers from the need to keep track of the details of superficially different representation systems. Since many *pkr* operations make no sense in domain models that are not frame-based, developers will need to be careful in using them within branches of code that are not specialized for frame-based representation systems.

18.8. Pfc -- The Reasoning Component

Pfc is a Prolog-based system which extends the reasoning capabilities of Prolog by providing integrated forward and backward chaining horn clauses, an integrated justification-based truth maintenance system (TMS), optional meta-level control of the inference strategy and the ability to easily intertwine Pfc and Prolog reasoning[10]. The addition of forward chaining to Prolog supports:

- The efficient computation of the deductive closure of a set of facts and rules that is necessary for many tasks such as classification, consistency checking, etc.
- The ability to support *blackboard*-type architectures for loosely coupled, cooperating systems.
- The ability to implement systems which acquire their information incrementally but must have a consistent interpretation at all times.

The addition of a TMS to Prolog supports default reasoning, explanations for deduced facts, and the ability to extend the reasoning of horn clauses to include counterfactuals and some disjunctive reasoning.

The current OPREPs version of PUNDIT uses Pfc for the template filling application. We are planning to use it to provide a general reasoning capability to enhance semantic interpretation and pragmatic analysis.

In the OPREPs template filling application, PUNDIT's output is added to the Pfc database at the end of each sentence in the message. As each fact is added to the database, inferences it enables are immediately drawn. These inferences can add additional facts to the database or invoke Prolog procedures. Pfc rules were written to enrich the IDR representation of events, deduce template fill information, select which template events to display and to actually manage the display as the message is processed.

For example, the Pfc rule in Figure 17 encodes the knowledge that if we don't know that an particular event's agent is friendly and we do know that its object is friendly, then we can infer that the event's agent is hostile. Note that the TMS will withdraw this inference whenever any new information, such as discovering that the agent is friendly, invalidates it.

As another example of a Pfc rule, consider the second rule in Figure 18. This rule has the following effect -- whenever a template event becomes known (by the assertion of a new *template_event* fact), the Prolog predicate *update_failsoft_rankings* is run taking as inputs the new event and its type.

14. CROSS-COMPONENT PHENOMENA

In a modular system such as PUNDIT, there are a number of interesting phenomena that require tight interaction across several components. We describe several of these below.

```
template_event(Situation,Type),
role(theme,Situation,Theme),
role(actor,Situation,Actor),
~friendly_C(Actor),
friendly_C(Theme)
   =>
template_info(Situation,2,hostile_C).

template_event(Sit,Type)
  =>
update_failsoft_rankings(Sit,Type).
```

Figure 17.
Sample Pfc Rule

14.1. Fragments

Sentence fragments provide a strong case for linguistically modular systems such as PUNDIT, because such elisions have distinct consequences at different levels of linguistic description. PUNDIT's approach to fragments can be summarized by saying that syntax detects 'holes' in surface structure and creates dummy elements as placeholders for the missing elements; semantics and pragmatics interpret these placeholders at the appropriate point in sentence processing, utilizing the same mechanisms for fragments as for full assertions[21].

Syntax regulates the holes

Fragment elisions cannot be accounted for in purely semantic/pragmatic terms. This is evidenced by the fact that there are syntactic restrictions on omissions; the acceptability of a sentence fragment hinges on grammatical factors rather than, e.g., how readily the elided material can be inferred from context.

Semantics and pragmatics fill the holes

In PUNDIT's treatment of fragments, each component contributes exactly what is appropriate to the specification of elided elements. Thus the syntax does not attempt to 'fill in' the holes that it discovers, unless that information is completely predictable given the structure at hand. Instead, it creates a dummy element. If the missing element is an elided subject, then the dummy element created by the syntactic component is assigned a referent by the pragmatics component. This referent is then assigned a thematic role by the semantics component like any other referent, and is subject to any selectional restrictions associated with the thematic role assigned to it. If the missing element is a verb, it is specified in either the syntactic or the semantic component, depending upon the fragment type.

Although the initial PUNDIT system was designed to handle full, as opposed to fragmentary, sentences, one of the interesting results of our work is that it has required only very minor changes to the system to handle the basic fragment types introduced below. These included the additions of: 6 fragment BNF definitions to the grammar (a 5% increase in grammar size) and 7 context-sensitive restrictions (a 12% increase in the number of restrictions); one semantic rule for the interpretation of the dummy element inserted for missing verbs; a minor modification to the reference resolution mechanism to treat elided noun phrases like pronouns; and a small addition to the temporal processing mechanism to handle tenseless fragments. The small number of changes to the semantic and pragmatic components reflects the fact that these components are not 'aware' that they are interpreting fragmentary structures, because the regularization performed by the syntactic component renders them structurally indistinguishable from full assertions.

14.2. Nominalizations

Syntactically, nominalizations are noun phrases, as in examples (1)-(7).

(1) An *inspection of lube oil filter* revealed metal particles.

(2) *Loss of lube oil pressure* occurred during *operation.*

(3) SAC received *high usage.*

(4) *Investigation* revealed adequate lube oil.

(5) Request *replacement of SAC.*

(6) *Erosion of impellor blade tip* is evident.

(7) Unit has low output air pressure, resulting in *slow gas turbine starts.*

Semantically, however, nominalizations resemble clauses, with a predicate/argument structure like that of the related verb. Our treatment attempts to capture these resemblances in such a way that very little machinery is needed to analyze nominalizations other than that already in place for other noun phrases and clauses.

There are two types of differences between the treatment of nominalizations and that of clauses. There are those based on *linguistic* differences, related to (1) the mapping between syntactic arguments and semantic roles, which is different in nominalizations and clauses, and (2) tense, which nominalizations lack. There are also differences in *control*; in particular, control of the filling of semantic roles and control of reference resolution. Finally, processing nominalizations requires coordinating syntactic, semantic, and pragmatic processing in such a way that each component makes the appropriate contribution to the analysis. All of these issues are discussed in detail in [6].

14.3. Semantic Raising

A further refinement of nominalization processing mechanism was added to the semantic interpreter in order to process RAINFORMS. This mechanism supports control and binding of implicit roles within nominalizations by the logical subject or object of the matrix clause. Among the verbs in the corpus which required this mechanism were *conduct*, *continue*, *cease*, *gain*, *hold*,*lose*, *sustain*, and *inflict*, as in:

(2-27) Both units...conducted gun attacks on Kobchik.
(subject is actor of *attack*)

(2-22) Loosefoot 722/723 continue search.
(subject is actor of *search*)

(2-57) ...sustained no damage from Kynda.
(elided subject is patient of *damage*)

This is similar to the phenomenon known as syntactic raising, but depends solely on the inherent semantics of the verb. Normally the only constituents available for mapping to predicate-arguments are the local constituents, so it

was necessary for the semantic interpreter to recognize during the processing of the matrix clause that its subject might be required during the processing of one of the other verb arguments. This allows it to be made available to the mapping process during the analysis of the verb argument. Thus, for example, PUNDIT is able to correctly represent the actor of the nominalization *searchR as the Loosefoots*, in the second example above. In order to invoke this type of processing for nominalizations, it is only necessary to declare it as part of the matrix verb's lexical semantics. Consequently, it is quite general, and has been applied in the TFR and OPREPS domains in addition to the RAINFORMS.

15. EVALUATION OF NATURAL LANGUAGE PROCESSING

Unisys has been a major contributor towards establishing evaluation criteria for natural language systems. During November, 1988, we organized and hosted the Natural Language Evaluation Workshop. We were also a key participant in the MUCK-II Message Understanding Conference. This section summarizes the current status of evaluation efforts at Unisys and in the natural language community at large.

15.1. Natural Language Evaluation Workshop

The Natural Language Evaluation Workshop brought together over 50 researchers in natural language processing and related fields. The workshop was sponsored by RADC, AAAI, and Unisys. It sought to to address the following basic questions:

What are valid measures of "black box" performance?
What linguistic theories are relevant to developing test suites?
How can we characterize efficiency?
What is a reasonable expectation for robustness?
What would constitute valid training sets and test sets?
How does all of this relate to measuring progress in the field?

Several concrete results came out of the workshop. In particular, a consensus was reached on the black box evaluation task for the second Message Understanding Conference, and a consensus was also reached on the desirability of a common corpus of annotated language, both written and spoken, that could be used for training and testing purposes. Since the workshop, the Message Understanding Conference has been held with interesting and useful results, and the Treebank project at the University of Pennsylvania has received funding and has begun. This should eventually lead to more formalized testing and comparisons of parsers. Evaluation is becoming a more prevalent topic at NL workshops, such at the one to be held at RADC in September of 1989, and the Darpa Spoken Language Community is working hard to construct a general evaluation procedure for the various contractors. However, most of the other specific workshops suggested, such as Data Base Question Answering, Generation, Knowledge Representation and Pragmatics and Discourse do not have any

funding sources available. The most difficult problems remain unresolved. We still do not know how to effectively measure improved performance during the crucial development phase, apart from peer review. We have little agreement on semantic representations, and there are still large classes of phenomena that have yet to be characterized in a scholarly fashion. However, this Workshop represented an important first step in building a consensus in the research community on evaluation methodologies for natural language systems.

15.2. MUCK-II

On June 6-8, the Language Understanding group participated in the second Message Understanding Conference (MUCK-II) at NOSC, where nine research groups demonstrated and evaluated their message understanding systems. For purposes of evaluation, each group addressed the same task on a common corpus of Navy OPREP messages; the task was to capture the significant events in each message by generating templates, e.g., an attack template, with slots for attacker, attackee, weapon, time and location. The conference included an on-site test on 5 new messages, plus reports of results on two other message sets, plus discussion of the underlying technology and evaluation procedures.

The MUCK-II conference included government participants from NOSC, DARPA, NIST, ONR, RADC, and other DoD agencies. A total of nine systems were represented at MUCK-II, including current DARPA contractors (NYU, SRI, Unisys), and also GE, PRC, GTE, ADS, MacDonnell-Douglas, and Language Systems.

The overall results of the conference were encouraging: message understanding technology has made major advances. The highest scoring systems reported scores of 80-90%, with good precision (90%) and reasonably high recall (80%). The Unisys system was the most ambitious in terms of linguistic coverage, but less highly tuned to the domain, producing good precision, but lower recall. Four of the systems were in early stages of development, and did not report results complete enough to compare to the other systems. One important finding (discussed in more detail below) was that different groups interpreted the scoring guide-lines quite differently. In addition, there were some reservations about what the score really represented, and as a result, there was agreement among the participants not to identify individual scores of participants. In keeping with that agreement, we will not discuss the actual scores in greater detail.

The value of the MUCK-II conference lay in having a diverse group of researchers address the same problem and in providing some objective means of comparing the effectiveness of different approaches. This serves two functions: evaluation of the maturity of message understanding technology and comparison of the effectiveness of different approaches. Thus choice of application and the associated evaluation procedure are issues central to the success of the conference. Probably more time was spent at the MUCK conference discussing

various aspects of evaluation and scoring than on any other single topic. The remainder of these comments will address some of the issues in evaluating message understanding systems:

1. How do we choose an appropriate task?
2. What do we want to evaluate?
3. How do we create a procedure that evaluates the right things?
4. How do we ensure a uniform interpretation of the evaluation procedure?
5. How do we ensure uniform scoring of the results?
6. What about other factors like level of effort, extensibility, etc.?"

How do we choose an appropriate task?

After the first MUCK conference, there was a feeling that we needed a specific task that could be objectively evaluated. The goal was to choose a task satisfying the following criteria, which evolved into the template fill task.

* Close to a possible "real" application,
* Required "understanding" of the message (not just keyword retrieval),
* Not so difficult as to be beyond the state of the art,
* Could be scored.

What do we want to evaluate?

There are at least two things that people wanted to evaluate:

* How well does the system perform the designated task?
* How well does it "understand" messages?

These goals are not necessarily consistent, however. The template fill task, for example, was sufficiently limited that is would have been possible to produce respectable results simply by exploiting holes in the scoring procedure and regularities of the template output. For example, of the 138 templates, 105 were "attack" templates. This raises the question of whether we want to allow clever ad hoc systems to complete with language understanding systems. In other words, do we want to evaluate task performance, or message understanding?

How do we create a procedure that evaluates the right things?

From the previous discussion, the right things are: 1) doing something useful (a "real" application) and 2) evaluating understanding. It is also important that the evaluation procedure distinguish "bad" mistakes from more harmless ones. For example, reversing the attacker/attackee relation in a template seems like a far more misleading error than getting the weapon type wrong (or even than getting the wrong template). From an informational point of view, it is also worse to get an answer wrong if there are only two choices (friendly vs. hostile) than if there are 10 choices. Similarly, if one answer occurs 90% of the time (e.g., the ATTACK template), it should be worse to get that answer wrong than to get an answer wrong which occurs only 10% of the time. Devising a scoring (or penalty)

procedure which considers the information content of answers may be one way to obtain a more accurate measure of "understanding", since it would penalize "guessing" more heavily. It should also provide a better metric of utility, if we can associate with each "answer" a cost of getting it wrong.

How do we ensure a uniform interpretation of the evaluation procedure?

Despite documentation and some pre-conference electronic mail discussions, it was clear that people arrived at (and even left) the conference with quite different interpretations of the evaluation procedure. One possible way to avoid this is to go through a pre-conference "debug" procedure, where training data results are submitted to a uniform scoring procedure (or a single, neutral scorer) and returned to the participants. This would have the advantage of debugging the scoring methods as well as debugging the participants' understanding of the scoring procedure.

How do we ensure uniform scoring of the results?

The resolution at MUCK-II was to turn over the scoring of the final test runs to a neutral party. However, this solution is very labor-intensive. A better solution is to have a scoring program that can automatically score results. This would have a number of advantages: first, it is clearly neutral (once the scoring procedure is agreed upon). Second, participants could set up their systems and have a neutral party run and evaluate the data. This would make it possible to keep the test data completely secure, as has been done in the speech community. Third, the scoring algorithm could be distributed to the participating groups, for use during development. This would be very beneficial, since it would provide a way of tracking progress during development: versions of the system can be tested and compared, in terms of overall scores.

What about other factors like level of effort, extensibility, etc.?

At MUCK-II, the only metrics were on accuracy of performance (score, precision, and recall). Participants were asked to report on total system development time and time spent on MUCK-II, but no attempt was made to factor these into the scoring procedure. Other performance metrics were not assessed at all, e.g., speed of the systems in doing the template fill task, ability to recover from errors, to debug the system, etc. All of these are important variables, and as systems mature, some of these will become discriminators between systems. It is important that we begin to think about how to measure these characteristics for future MUCK conferences.

The MUCK-II conference demonstrated that message understanding is maturing rapidly, and that operational systems for limited domains may only be a few years away. Even a year ago, many participants were extremely skeptical about the possibility of evaluating natural language systems in any meaningful way and were also skeptical of building systems that could perform respectably on some kind of "task". These people are now talking about the design of MUCK-III. The challenge for MUCK-III is not only to produce better systems, but to

make sure that we choose applications and evaluate attributes that will b relevant to high performance in real applications.

15.3. Evaluating Parsing in the Resource Management Domain

In order to evaluate the coverage of PUNDIT, we ported PUNDIT to the Resource Management domain, a set of 791 training sentences consisting of queries to a Navy database about ships, their capabilities and their locations. This represented our first test of PUNDIT on a query corpus[18]; its previous applications had been to messages and abstracts. Our methodology was to increase coverage of PUNDIT as necessary, to cover the Resource Management training sentences. We thus added (only a few) domain-independent rules to the grammar, and domain-independent entries to the lexicon, to cover the major constructions observed in the Resource Management corpus. We then trained on a (subset of) this corpus. The training involved parsing the first 200 sentences and examining and fixing parsing problems in these 200 sentences. We were able to collect selectional patterns only for the first 100 sentences. In both cases, this represents only a small fraction of the available training data (791 sentences). The sentences (training and test) were run on PUNDIT, under Quintus Prolog 2.2 on a Sun 3/60 with 8 MB of memory.

Because PUNDIT normally produces many parses, especially when run without selectional constraints, we allowed the system to run to a maximum of 15 parses per sentence. We report several results below, for purposes of comparison with other groups presenting parsing results. The first result is the number of sentences *obtaining a parse*. We believe that this is not a meaningful figure, however, since it is possible for a sentence to obtain a parse, but never to obtain a *correct* parse. For this reason, we report a second result: the number of sentences obtaining a *correct parse* within the first 15 parses. In some cases, the system obtained a parse, but did NOT obtain the correct parse within the first 15 parses. In this case, we report it as NOT GETTING A CORRECT PARSE.

Our criteria for counting a parse correct were generally very stringent, and also required obtaining the correct regularized syntactic expression (or ISR). Our criteria included, for example: correct scoping of modifiers under conjunction; correct attachment of prepositional phrase and relative clause modifiers; and correct analysis of complex verb objects.

The table below (Figure 18) shows the results obtained with parsing alone (no selectional constraints). We did not report the results obtained from selection, because it turned out that, given our very limited collection of patterns, selection failed to change the test results significantly. However, we plan to collect patterns for the entire training set and rerun this portion of the experiment.

There are several things worth noting in these results. First, the system is quite fast, even running to 15 parses: the average parse time to the correct parse is under 10 seconds for sentences averaging about 10 words/sentence. Second, although the correct parse appears on the average in the third parse, the first

	Training (200 sentences)	Test (200 sentences)
Get A PARSE	94%	92%
Get A CORRECT PARSE using SYNTAX only	85%	76%
avg. # of correct parse	2.9	2.6
avg. # of parses/sentence	7.1	6.2
avg. secs. to correct parse	7.5	4.9
avg. secs. total	25.5	17.8

Figure 18.
Parsing Results for the Resource Management Domain

parse is correct more than 40% of that time. By adding semantic constraints, we expect to improve that figure substantially, thus driving down further the time to obtain the correct parse.

As stated above, these results were obtained without the use of domain-specific semantic selectional patterns. Since parsing results improve significantly when these patterns are available, we are very optimistic that PUNDIT's coverage is indeed adequate to a range of tasks, from query processing to message understanding.

16. PARALLELISM

Parsing is a search problem -- there is a great deal of non-determinism in parsing, especially given a large grammar. This stems from two sources: first, the need to explore paths that end in failure. For example, given the initial string *the bus stops*, we have no idea if we have a noun phrase (*the bus stops are indicated in red*) or subject + verb (*the bus stops here*). Second, there may be ambiguity, particularly in the absence of domain-specific knowledge, which results in multiple parses for a given sentence, e.g., *starting air compressor failed = starting the air compressor failed* or *the starting air compressor failed.* The correct parse depends on domain information. The search space explosion in a large grammar, particularly when dealing with fragmentary input or long sentences, makes parsing an ideal application for exploiting or-parallel search.

Over the past two years, we have conducted a set of experiments that indica that substantial speed-ups can be obtained by exploiting or-parallelism at th level of grammar rule disjunctions in RG. Our evidence comes from two sources. First, we ran a series of simulation experiments assuming a shared-memory multi-processor architecture running an or-parallel Prolog[16, 19]. Second, we have been able to run a preliminary series of experiments on an actual or-parallel Prolog implementation, which confirmed the availability of useful or-parallelism during parsing.

In the simulation experiments, we searched for all solutions (parses) in parallel. One of the attractive features of this paradigm is that the search paths can be pursued independently of one another: when a grammar rule has a disjunction, the processing simply splits: one processor pursues the parse, applying the first disjunct, and runs until completion or failure; the other processor does likewise, except that it must copy the current "state of the parse", naming all logic variables apart, before initiating execution of the second disjunct. On failure, a processor simply returns to the processor pool, and is assigned a new parse state and rule disjunction to pursue.

This simple approach to spawning or-parallel processes turned out to be sufficient to obtain expected speed-ups of 20-30 fold, running to all parses for a set of sentences from Navy messages [16]. Our initial concern was that the copying costs to copy a given "parse state" might be prohibitive. Experimentation showed that the key factor in generating useful or-parallelism was the ratio between process granularity (the median process duration between process spawns) to the time for process start-up (including the time to copy the parse state). We can rephrase this as how much time is spent spent doing useful work in parallel vs. the time spent in overhead, setting up the parallel processes. It turns out that if this ratio goes below 2:1, more than half the available speed-up is lost.

RG has several properties that made it possible to exploit the parallelism effectively. First, the parse state to be copied turns out to be the *path* from the node under construction to the root. This is where the tree is "growing", that is, where variables will be instantiated as the parse continues. Second, because of restrictions and automatic tree building, the granularity of the processes turned out to be big enough relative to the estimated copying cost: our simulation results showed an average process duration of 9 ms., vs. an estimated copying time of 3 ms., to give a 3:1 ratio, enough to make effective use of the available parallelism.

These results were confirmed by runs on the Aurora Prolog system, the or-parallel Prolog system under development at the Swedish Institute for Computer Science (SICS)[13]. We achieved average speed-ups of greater than 10-fold using only 12 processors. For several of the tested sentences, these results exceeded the maximum predicted speed-ups given by the simulation experiments. Furthermore, we have reason to be very optimistic that the

performance will become even better. We measured the speed-ups using 4, 6, 8, 10, and 12 processors, and there did not seem to be any reduction in the speed-up improvement as more processors were added, indicating that we had not yet exhausted the available parallelism.

We are now in the process of planning a series of further experiments, both to relate these results to our earlier simulation experiments, and to increase the number of processors.

17. IMPLEMENTATION

PUNDIT is implemented in Edinburgh-style Prolog. It is currently implemented in Quintus Prolog, but we have ported in to a number of other Prologs, including Explorer-Prolog, C-Prolog, and SICSTUS Prolog. Because we have ported the system to a variety of Prolog's, we have been careful to factor out implementation-specific features, to keep the system easy to port.

The core (domain-independent) portion of PUNDIT consists of some 20,000 lines of code. The domain-specific modules (lexicon, knowledge base, semantics rules, selection patterns) add an additional 20-70% (5,000 to 15,000 lines, depending on the application). In addition, we have buil l a number of tools, many of which are not included in the "core" system.

In terms of performance, a PUNDIT image occupies several megabytes of space. Running on a SUN 3/60 with 8 MB of memory, it takes approximately 1-3 seconds to parse an 8-10 word sentence. As the input becomes more telegraphic, or longer, or malformed (e.g., a run-on), it may take substantially longer. A typical paragraph takes between 10-30 seconds to process, depending again on its complexity and the amount of "back-end" application specific processing required. We have found these speeds to be quite reasonable for development work. Additional memory and/or a faster CPU will obviously increase the speed significantly.

18. FUTURE PLANS

PUNDIT represents the most ambitious attempt to address the understanding of written text, based on linguistic principles. It is unique in integrating, in modular fashion, input from syntax, semantics, pragmatics, domain knowledge, and discourse structure, to produce a single representation of information in the text. It has been successfully applied to a range of applications, including message summarization, database creation, database query, and a high-level interface for expert systems. It also forms the basis for our current work in Spoken Language systems.

Because of the breadth of its scope, there are many gaps in PUNDIT's coverage. A partial list includes:

A much tighter coupling between syntax and semantics;
A treatment of scope for quantifiers and negation;
A treatment of "non-real" events (future, imperatives, modals, negated assertions);

A treatment of event reference;
A detailed treatment of intersentential time relations;
A domain-independent lexicon; and
A tight coupling between the reasoning component and semantics and pragmatics.

In addition, for spoken language, we must address issues of constraining and training the syntax and semantics to a new domain; preventing overgeneration in the grammar; and recovering from disfluencies (false starts, interrupts). To this end, we have implemented several experimental modules, including a module for semantics-based selection, a bottom-up parser, and the use of a well-formed substring table for better top-down parsing. All of these modules are under development, and have not yet been integrated into the "stable" PUNDIT system.

Although there are many areas not yet covered in PUNDIT, we believe that the PUNDIT system has made an important contribution to natural language understanding research, in demonstrating the feasibility of building of a modular, portable, linguistically-based system for language understanding. We look forward to continuing to extend this work into new message domains, to other kinds of input (journal articles, spoken language), and even to other languages.

19. ACKNOWLEDGEMENTS

The construction of PUNDIT represents a team effort: over the four years of the contract, many people have played key roles in the development of the system. We would like to acknowledge their contributions here. First, the development of the semantic/pragmatic processing has been led by Martha Palmer (semantics component) with key contributions from Deborah Dahl (reference resolution) and Rebecca Passonneau (Time), and with more recent contributions from Catherine Ball (interactive discourse component) and Carl Weir (integration into PUNDIT of the knowledge base and reasoning component), and also Bonnie Webber, who joined us for a year, part-time to work on issues of temporal and event reference. The development of the syntactic and selectional components has been led by Lynette Hirschman, with key contributions from John Dowding (parser, regularized syntactic representation), Marcia Linebarger (syntax, lexicon), and Francois Lang (selection). In addition, other people who are no longer with the project have made important contributions: Leslie Riley (tool development), and Korrinn Fu (system administration procedure). We also welcome two recent additions to the group, Lewis Norton (knowledge representation) and Shirley Steele (speech interfaces). In addition, the Knowledge Management Group at Paoli Research Center, led by Tim Finin, has made a substantial contribution by providing a knowledge representation framework (M-Pack, David Matuszek), the Prolog forward chaining system (Tim Finin, Richard Fritzson), and tools and interfaces (Dan Corpron). The knowledge representation work is based on earlier work by Michael Freeman, whose untimely death deprived us of a valued colleague and friend; we wish to acknowledge here Mike's contribution

to this work, and to Paoli Research Center as a whole. In addition, we would like to acknowledge the support and encouragement we received from PRC Directors Earl Riegel, Bruce Roberts and Allen Sears. We would also like to thank Millie Miele for her expert assistance in the production of this report.

We would also like to acknowledge our colleagues at NYU, led by Ralph Grishman, particularly Mark Gawron, N.T. Nhan and Tomas Ksiezyk, who shared many of their insights with us, especially in the first two years of the project. We would also like to thank Beth Sundheim at NOSC for organizing the first and second message understanding conferences, which have been crucial to the development of the field. Finally, we would like to acknowledge our Program Managers at DARPA, who have contributed their insights and support to make this a success: Ron Ohlander, Bob Simpson, Allen Sears, and Charles Wayne.

20. BIBLIOGRPAHY FOR PUNDIT SYSTEM

(1) Ball, C., Dahl, D., and Hirschman, L. "Answers and Questions: Processing Messages and Queries. To be presented at the Darpa Spoken Language Meeting Harwich Port, MA, October, 1989.

(2) Ball, Catherine N. "Natural Language Processing: An Overview". USE, Inc. Conference, Montreal, October, 15, 1987.

(3) Ball, C., Dowding, J., Lang, F., and Weir, C. "PUNDIT User's Guide". LBS Technical report, June 1988.

(4) Ball, C. "Analyzing explicitly-structured discourse in a limited domain: trouble and failure reports". In *Proceedings of the DARPA Workshop on Speech and Language*, March 1989.

(5) Dahl, D. *The Structure and Function of* one-*Anaphora in English.* Bloomington, Indiana, Indiana University Linguistics Club, October, 1985.

(6) Dahl, D. "Directions in Natural Language Interaction with Computers". Presented at the Fall Meeting of the Institute of Industrial Engineers. Chicago, December, 1985.

(7) Dahl, D. "Focusing and Reference Resolution in PUNDIT". AAAI-86, Philadelphia, August, 1986.

(8) Dahl, Deborah A. "Determiners, Entities, and Contexts". *Proceedings of TINLAP-3, Theoretical Issues in Natural Language Processing.* Yorick Wilkes, ed., January, 1987.

(9) Dahl, Deborah A., Palmer, Martha S., and Passonneau, Rebecca. "Nominalizations in PUNDIT". *Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics*, Stanford, CA., July, 1987.

(10) Dahl, Deborah A. "Integration of Semantics and Pragmatics in the Computational Analysis of Nominalizations". Colloquium presented to the Department of Computer Science, The Pennsylvania State University, October, 1987.

(11) Dahl, Deborah, John Dowding, Lynette Hirschman, Francois Lang, Marcia Linebarger, Martha Palmer, Rebecca Passonneau, Leslie Riley, "Integrating Syntax, Semantics, and Discourse". Darpa Natural Language Understanding Program. R&D Status Report Unisys Defense Systems, May 14, 1987.

(12) Dahl, D. "Natural Language Understanding for Database Generation: The PUNDIT System", Invited talk presented at AI West, Long Beach, California, May, 1988.

(13) Dahl, D., Hirschman, L., and Ball, C. "A Black Box Evaluation of PUNDIT", Presented at the Workshop on Evaluation of Natural Language Systems, December 8-9, 1988.

(14) Dahl, D., and Ball, C. "Reference Resolution in PUNDIT", to appear in *Logic and Logic Grammars for Language Processing,* edited by Stan

Szpakowicz and Patrick Saint-Dizier, Ellis Horwood, LTD.

(15) Dowding, John, and Hirschman, Lynette. "A Dynamic Translator for Rule Pruning in Restriction Grammar". *Natural Language Understanding and Logic Programming, Vol. II*, eds. V. Dahl and P. Saint-Dizier, North-Holland, Amsterdam, 1988, pp. 79-92.

(16) Dowding, J. "Reducing Search by Partitioning the Word Network". In *Proceedings of the DARPA Workshop on Speech and Language*, March 1989.

(17) Dowding, J. "A continuation based parser for restrictions". Presented at the Workshop on Logic Programming and Natural Language Processing, Stockholm, Sweden, April 1989.

(18) Finin, Tim, Richard Fritzson and David Matuszek. "Adding Forward Chaining and Truth Maintenance to Prolog". In *Proceedings of the Fifth Conference on Artificial Intelligence Applications,* March 1989.

(17) Grishman, R., Hirschman, L., and Nhan, Ngo Thanh. "Discovery Procedures for Sublanguage Selectional Patterns: Initial Experiments", *Computational Linguistics,* Volume 12, No. 3, pp. 205-215.

(19) Hirschman, L. Panelist for Panel Discussion on Logic-Based Meta-Grammars (w. V. Dahl, F. Pereira, H. Abramson), Workshop on Theoretical Approaches to Natural Language Understanding, Halifax, Nova Scotia, May, 1985.

(20) Hirschman, L., and Puder, K. "Restriction Grammar: A Prolog Implementation", in *Logic Programming and its Applications*, D.H.D. Warren and M. VanCaneghem, eds., December, 1985.

(21) Hirschman, L. "Meta-Conjunction in Restriction Grammar." *Journal of Logic Programming.*

(22) Hirschman, L., "AI and DB Research at SDC: An Overview", RADC Workshop on AI and DB, Minnowbrook, NY, July, 1986.

(23) Hirschman, L., "Natural Language Interfaces to Databases Revisited", RADC Workshop on AI and DB, Minnowbrook, NY, July, 1986.

(24) Hirschman, L., and Grishman, R. "PROTEUS and PUNDIT: Research in Text Understanding", *Computational Linguistics,* Volume 12, No. 2, April-June, 1986, pp. 141-145.

(25) Hirschman, Lynette. "Natural Language Interfaces for Large Scale Information Processing". In *Integration of Information Systems: Bridging Heterogeneous Databases*, ed. A. Gupta, IEEE Press, New York, 1989, pp. 308-314.

(26) Hirschman, Lynette, "Tutorial on Natural Language and Logic Programming". 1987 Logic Programming Symposium, San Francisco, Aug. 31-Sept. 4, 1987.

(27) Hirschman, Lynette Deborah Dahl, John Dowding. Francois Lang, Marcia Linebarger, Martha Palmer. Leslie Rile, Rebecca [illegible]assonneau] Schiffman,

"The PUNDIT Natural Language Processing System". Presented at the Eleventh Annual Penn Linguistics Colloquium, Philadelphia, PA, February, 1987.

(28) Hirschman, L. "A Meta-Treatment of Wh-Constructions". *META88: Proceedings of the Workshop on Meta-Programming in Logic Programming,* June, 1988.

(29) Hirschman, L., Hopkins, W.C., Smith, R.C. "Or-Parallel Speed-up in Natural Language Processing: A Case Study". *Proc. of the 5th International Logic Programming Conference*, eds. R. Kowalski and K. Bowen, Seattle, August, 1988, pp. 263-279.

(30) Hirschman, L., M. Palmer, J. Dowding, D. Dahl, M. Linebarger, R. Passonneau, F-M. Lang, C. Ball, and C. Weir. "The Pundit Natural Language System", In *Proceedings of the Conference on Artificial Intelligence in Government*, Washington, D.C. March, 1989, pp. 234-243.

(31) Hirschman, L., F-M Lang, J. Dowding and C. Weir. "Porting PUNDIT to the Resource Management Domain". In *Proceedings of the DARPA Workshop on Speech and Natural Language*, March 1989.

(32) Hirschman, L. "Computational Requirements for Spoken Language Systems". Presented at the Workshop on Logic Programming and Natural Language Processing, Stockholm, Sweden, April 1989.

(33) Hirschman, L., "Computational Requirements for a Spoken Language System". To be presented at the Darpa contractors' meeting, October, 1989.

(34) Hirschman, L., and Dowding, J. "Restriction Grammar: A Logic Grammar", to appear in *Logic and Logic Grammars for Language Processing,*, edited by Stan Szpakowicz and Patrick St. Dizier, Ellis Horwood, LTD.

(35) Hopkins, W., Hirschman, L., and Smith, R. "Or-Parallelism in Natural Language Parsing", to appear in *Parallel Algorithms for Machine Intelligence and Pattern Recognition*, eds., V. Kuman, P.S. Gopalkrishman, and L. Kanal, Springer-Verlag, New York, 1990.

(36) Lang, Francois-Michel. Intersentential Cancellation of Scalar Implicature: In Theory and In Fact. Presented at the Eleventh Annual Penn Linguistics Colloquium, Philadelphia, PA, February, 1987.

(37) Lang, Francois. "A User's Guide to the Selection Module". LBS Technical Memo No. 68. Paoli Research Center, 1987.

(38) Lang, Francois, and Hirschman, Lynette. "Improved Parsing through Interactive Acquisition of Selectional Patterns". LBS Technical Memo No. 69, Paoli Research Center, 1987.

(39) Lang, F. "The PUNDIT Text-Understanding System: Approaches to Domain Portability". Invited talk presented at Carnegie-Mellon University, May 25, 1988.

(40) Lang, F., and Hirschman, L."Improved Portability and Parsing Through Interactive Acquisition of Semantic Information". *Proceedings of the Second Conference on Applied Natural Language Processing*, Austin, Texas, February, 1988.

(41) Lang, Francois. "Pundit's First French Lesson: The PRATTFALL Machine-Translation Module". Presented at the Penn Linguistics Colloquium, February 17-18, 1989, and at the PRC AI Seminar, February 15, 1989.

(42) Linebarger, M. "Neuropsychological Evidence for Linguistic Modularity." in *Studies in Theoretical Psycholinguistics*, G. Carlson and M. Tanenhaus, eds. (to appear).

(43) Linebarger, M. "Neuropsychology of Sentence Parsing." in *Advances in Cognitive Neuropsychology.* (to appear).

(44) Linebarger, Marcia C. "Negative Polarity and Grammatical Representation". *Linguistics and Philosphy.* Vol. 10. no. 3, August, 1987.

(45) Linebarger, M. "A Guide to Object Options in PUNDIT". LBS Technical Report, July, 1988.

(46) Linebarger, M., Dahl, D., Hirschman, L., and Passonneau, R. "Sentence Fragments Regular Structures". *Proceedings of the 26th Annual Meeting of the Association for Computational Linguistics.* Buffalo, June, 1988.

(47) Palmer, M. *Driving Semantics for a Limited Domain.* Ph.D. thesis, University of Edinburgh. July, 1985.

(48) Palmer, M., Dahl, D., Schiffman, R., Hirschman, L., Linebarger, M., and Dowding, J. "Recovering Implicit Information", *Proceedings of the 24th Meeting of the Association for Computational Linguistics*, June, 1986.

(49) Palmer, Martha S. "Natural Language Series," One of the Artificial Intelligence Short Courses (7 lectures) Televised at the University of Maryland, Instructional Television System, January, 1987.

(50) Palmer, Martha. "Developing and Porting a Text Processor". Invited talk presented at Bell Labs, December 11, 1987.

(51) Palmer, Martha S. *Semantic Processing for Finite Domains,* to appear as a volume in Studies in Natural Language Processing, Cambridge University Press, editor, Aravind Joshi, 1988.

(52) Palmer, M., and Linebarger, M. "Status of Verb Representations in PUNDIT". Presented at Theoretical And Computational Issues in Lexical Semantics, Brandeis University, Waltham, Mass, April 21-24, 1988.

(53) Passonneau, Rebecca. "Situations and Intervals". *Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics*, Stanford, CA., July, 1987.

(54) Passonneau, R. "A Computational Model of the Semantics of Tense and Aspect". *Computational Linguistics,* vol. 14, no. 2, June, 1988.

(55) Passonneau, R. "Getting at Discourse Referents". *Proc. of the 27th Annual Meeting of the Association for Computational Linguistics,* Vancouver, June, 1989.

(56) Passonneau, R., Weir, C., and Finin, T. "Interfacing Natural Language Processing and Knowledge-based Processing in PUNDIT". To be presented at the Darpa contractors' meeting, October, 1989.

(57) Pierrehumbert, Janet, and Steele, Shirley. "Categories of Tonal Alignment in English". To appear in *Phonetica.*

(58) Riley, L. "A Guide to the PUNDIT Lexical Entry Procedure". LBS Technical Report, July, 1988.

(59) Schwartz, M., Linebarger, M., Saffran, E., & D.S. Pate. Syntactic transparency and sentence interpretation in aphasia. *Language and Cognitive Processes,* Vol. 2, no. 2, pp 85-113.

(60) Sproat, Richard, and Steele, Shirley. "An Investigation of Tag Intonation in English". Presented at the November Meeting of the Acoustical Society of America.

(61) Webber, Bonnie L. "Position Paper: Event Reference". *Proceedings of TINLAP-3, Theoretical Issues in Natural Language Processing.* Yorick Wilkes, ed., January, 1987.

(62) Webber, Bonnie L. "The Interpretation of Tense in Discourse". Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics, Stanford, CA., July, 1987.

(63) Weir, C. "Semantic Properties of English Gerundives". Invited talk presented to LiLog KI-Kolloquium, IBM Germany, Stuttgart, Germany, September 16, 1988.

21. REFERENCES

[1] Strategic Computing -- New Generation Computing Technology:, A Strategic Plan for its Development and Application to Critical Problems in Defense, Defense Advanced Research Projects Agency, October, 1983.

[2] C. Ball, Analysing explicitly-structured discourse in a limited domain: trouble and failure reports. In *Proceedings of the DARPA Workshop on Speech and Language*, March 1989.

[3] R. Brachman and J. G. Schmolze, An Overview of the KL-ONE Knowledge Representation System. *Cognitive Science* 9, 1985, pp. 171-261.

[4] D. Dahl and C. Ball, Reference Resolution in PUNDIT. In *Logic and Logic Grammars for Language Processing*, S. Szpakowicz and P. Saint-Dizier (ed.), Ellis Horwood, 1990.

[5] Deborah A. Dahl, Focusing and Reference Resolution in PUNDIT, Presented at AAAI, Philadelphia, PA, 1986.

[6] Deborah A. Dahl, Martha S. Palmer, and Rebecca J. Passonneau, Nominalizations in PUNDIT, Proceedings of the '5th Annual Meeting of the ACL, Stanford, CA, July, 1987.

[7] Deborah A. Dahl, John Dowding, Lynette Hirschman, Francois Lang, Marcia Linebarger, Martha Palmer, Rebecca Passonneau, and Leslie Riley, Integrating Syntax, Semantics, and Discourse: DARPA Natural Language Understanding Program, R&D Status Report, Paoli Research Center, Unisys Defense Systems, May 14, 1987.

[8] John Dowding and Lynette Hirschman, Dynamic Translation for Rule Pruning in Restriction Grammar, Presen[Bted at the 2nd International Workshop on Natural Language Understanding and Logic Programming, Vancouver, B.C., Canada, 1987.

[9] J. Dowding, A continuation based parser for restrictions, Presented at the Workshop on Logic Programming and Natural Language Processing , Stockholm, Sweden, April 1989.

[10] T. Finin, R. Fritsson, and D. Matussek, Adding Forward Chaining and Truth Maintenance to Prolog. In *Proc. of the 5th Conference on Artificial Intelligence*, March 1989.

[11] Michael W. Freeman, *KNET: An Extended SI-Net Formalism for Knowledge Representation Systems.* TR [illegible]M-80.1, ADO/FSSG, Burroughs Corporation, Paoli, PA, January, 1980.

[12] Z. Harris, *String Analysis of Sentence Structure*, The Hague, 1962.

[13] B. Hausman, A. Ciepielewski, and S. Haridi, OR-Parallel Prolog Made Efficient on Shared Memory Multiprocessors. In *Proc. of the 1987 Symposium on Logic Programming*, San Francisco, 1987, pp. 69-79.

[14] L. Hirschman and K. Puder, Restriction Grammar: A Prolog Implementation. In *Logic Programming and its Applications*, D.H.D. Warren and M. VanCaneghem (ed.), Ablex Publishing Corp., Norwood, N.J., 1986, pp. 244-261.

[15] L. Hirschman, Conjunction in Meta-Restriction Grammar. *J. of Logic Programming* 3(4), 1986, pp. 299-328.

[16] L. Hirschman, W. Hopkins, and R. Smith, Or-Parallel Speed-up in Natural Language Processing: A Case Study. In *Proc. of the 5th International Logic Programming Conference*, R. Kowalski and K. Bowen (ed.), Seattle, August 1988, pp. 263-279.

[17] L. Hirschman, A Meta-Treatment of Wh-Constructions. In *Meta88: Proc. of the Workshop on Meta-Programming in Logic Programming*, Bristol, U.K., June 1988.

[18] L. Hirschman, F. Lang, J. Dowding, and C. Weir, Porting PUNDIT to the Resource Management Domain. In *Proceedings of the DARPA Workshop on Speech and Natural Language*, March 1989.

[19] W. Hopkins, L. Hirschman, and R. Smith, Or-Parallelism in Natural Language Parsing. In *Parallel Algorithms for Machine Intelligence and Pattern Recognition*, V. Kuman, P.S. Gopalkrishman, and L. Kanal (ed.), Spring-Verlag, New York, 1990.

[20] F. Lang and L. Hirschman, Improved Portability and Parsing Through Interactive Acquisition of Semantic Information. In *Proceedings of the Second Conference on Applied Natural Language Processing*, Austin, TX, February 1988.

[21] M. C. Linebarger, D. A. Dahl, L. Hirschman, and R. J. Passonneau, Sentence Fragments Regular Structures. *Proc. of the 26th ACL Conference*, June, 1988.

[22] D. Matuszek, A Programmer's Interface to KNET, Technical Memo 61, Paoli Reseach Center, Unisys Corporation, October 1987.

[23] Martha S. Palmer, Driving Semantics for a Limited Domain, Ph.D. thesis, University of Edinburgh, 1985.

[24] Martha S. Palmer, Deborah A. Dahl, Rebecca J. [Passonneau] Schiffman, Lynette Hirschman, Marcia Linebarger, and John Dowding, Recovering Implicit Information, Presented at the 24th Annual Meeting of the Association for Computational Linguistics, Columbia University, New York, August 1986.

[25] Rebecca J. Passonneau, Situations and Intervals, Presented at the 25th Annual Meeting of the Association for Computational Linguistics, Stanford University, California, July 1987.

[26] Rebecca J. Passonneau, A Computational Model of the Semantics of Tense and Aspect. *Computational Linguistics* 14(2), June 1988, pp. 44-60.